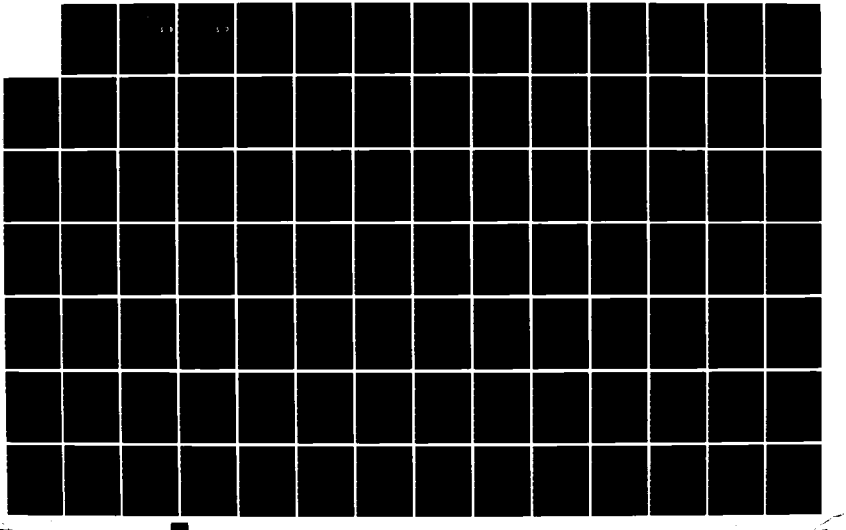
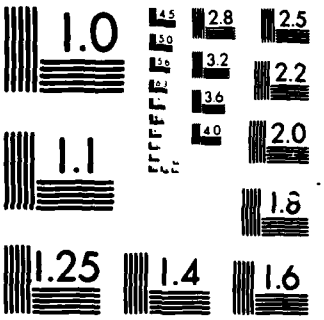
AD-A163 843 DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY FOR A DISTRIBUT..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J A WEDERTZ 1/2

UNCLASSIFIED DEC 85 AFIT/GCS/ENG/85D-24 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963-A

DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY FOR A DISTRIBUTED DATABASE MANAGEMENT SYSTEM

THESIS

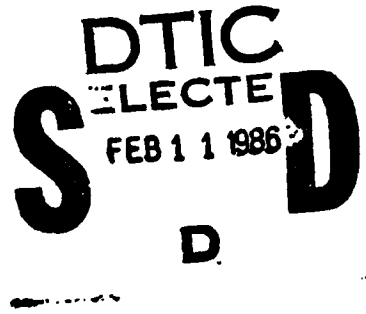
James A. Wedertz
Captain, USAF

AFIT/GCS/ENG/85D-24

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/85



DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY FOR A DISTRIBUTED DATABASE MANAGEMENT SYSTEM

THESIS

James A. Wedertz
Captain, USAF

AFIT/GCS/ENG/85D-24

Approved for public release; distribution unlimited

AFIT/GCS/ENG/85D-24

DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY FOR A DISTRIBUTED DATABASE MANAGEMENT SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

James A. Wedertz, B. S.
Captain, USAF

December 1985

Preface

The purpose of this study was to design and implement a centralized data directory for a distributed database management system being developed at the AFIT Digital Engineering Laboratory. The first phase of this project included a requirements analysis documented in Structured Analysis Design Technique (SADT) diagrams. In the next step, structure charts showed the detailed design of the software module hierarchy. During the following phase, part of the design involving the central directory was implemented on two microcomputers in the laboratory. Finally, this study discussed the project results and recommendations for future studies.

During the entire process of this study, I received a lot of help from many people. I am very grateful to my thesis adviser, Dr. Thomas C. Hartrum, for his suggestions and analytical skill in resolving many computer interface problems. Also, I am thankful to Major Walter Seward, another member of my thesis committee, for reviewing this work. My thanks also to Mr. Charlie Powers and Mr. Dan Zambon for their invaluable technical support in the laboratory. Finally, I am extremely grateful to my dear wife, Bettylou, for typing the many drafts of this thesis and assuming many of my household responsibilities during this project. Without her loving support, I never would have completed this thesis.

James A. Wedertz

Table of Contents

*These appendices are in an additional thesis volume maintained at AFIT/ENG: Volume II: DDBMS Current Implementation

List of Figures

Abstract

This study refined and implemented a design of a centralized data directory for a distributed database management system (DDBMS) begun in a previous study for use in the AFIT Digital Engineering Laboratory. This directory contains information about all the data stored in the distributed databases. By following the life cycle programming method to develop the system, this project completed a requirements analysis, detailed design and implementation of the data directory as well as a partial implementation of the DDBMS to test the operation of the centralized data directory.

The requirements analysis outlined the functions of the central site, which contained the centralized directory. This project used Structured Analysis Design Technique (SADT) diagrams to document the central site's functions. These included initializing the DDBMS, updating the central directory, sending changes to other local directories at the remote sites, reconfiguring the DDBMS and servicing requests for information in the directory.

Next, the project refined the detailed design of the CNDD processing and depicted the functional decomposition in structure charts. The following step implemented on two microcomputers only those modules necessary to show the centralized directory worked. Tests verified that one DDBMS

node which received a query could request and receive location information from the other node.

DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY SYSTEM FOR A DISTRIBUTED DATABASE MANAGEMENT SYSTEM

I. INTRODUCTION

Background

Many organizations store the data used in their various computer programs in a database. This allows them to centralize the information so that it is easier to retrieve and change the data. A centralized database management system (DBMS) consists of software residing on one computer which structures the data and manipulates it so that many application programs can access it. On the other hand, a distributed database management system (DDBMS) manipulates separate databases stored in host computers which are linked in a network. The network connects host computers either dispersed over a short distance (usually less than 1 km) as in a local area network (LAN) or geographically distributed over a large area as in a long haul network (Tanenbaum, 1981:4-5). However, distribution is transparent to the user so he can access any data in the system without having to know where it is stored.

Organizations develop distributed databases for several reasons as Ceri & Pelagatti pointed out (Ceri, 1984:11-12). First, they may use such a system for organizational and

economic reasons. If the organization has decentralized operations, the DDBMS may fit the structure more naturally. Also, large mainframe computers installed at a central location may not be as economical as dispersed smaller computers.

Second, organizations may want to connect existing databases rather than create a new one to support new applications. Third, distributed databases allow for incremental growth. Adding a new database on the system should have limited impact on the existing databases.

Fourth, distributed databases can increase performance. The load can be shared among processors to allow parallel operations. Also, if each processor can do its operations alone without interfering with another processor, there will be less communications congestion.

The fifth and final reason Ceri and Pelagatti mentioned for using a distributed database is to increase system reliability. If one system goes down, this only affects the applications at that site and those that use the data stored there. In other words, one system failure should not cause the entire system to crash.

Besides these advantages for developing a distributed database, Capt. John G. Boeckman explained in his thesis some disadvantages (Boeckman, 1984:2). The DDBMS is more complex than a centralized DBMS. It must interface with a network in order to reliably send and receive data. Also, queries--requests for information--must be decomposed effi-

ciently because the data may be stored in many places. In addition, deadlock may occur when two or more systems are waiting to update data held by the other system.

Data concurrency--or keeping several copies of the same data current--is another problem with distributed databases. The DDBMS must use complex algorithms to synchronize these data updates. In addition, a DDBMS must maintain a conceptual--or overall--view of all the data in the system. This requires a data dictionary/directory to keep track of data locations, among other things.

According to Peebles and Manning, there are three approaches to designing a DDBMS architecture (Peebles, 1979:351-357): integrated, homogeneous, and heterogeneous. The following summarizes the differences between them.

In the integrated model (Figure 1a), each DBMS connects directly to the network and can access information in another DBMS without translating data. Reducing the useful CPU time and the amount of memory needed for the data exchange process are two advantages of this architecture.

In the homogeneous model (Figure 1b), each computer in the network supports the same DBMS (e.g. INGRES). Each computer, however, has separate DBMS and communication software modules. The latter module performs the data exchange functions.

In the heterogeneous model (Figure 1c), the computers can support different types of DBMS. For example, both

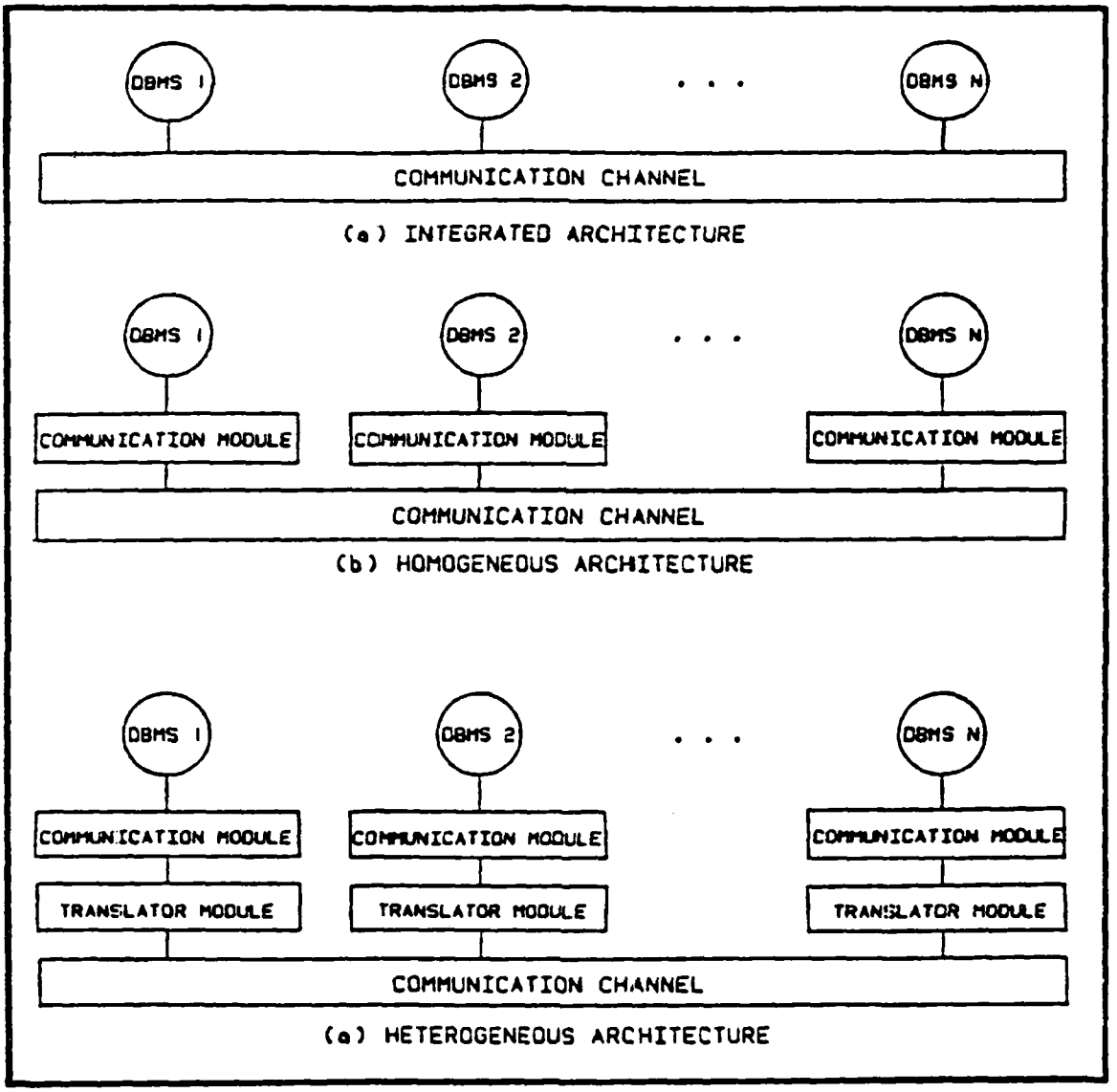


Figure 1. DDBMS Architectures

INGRES, a relational-type DBMS, and Total, a network-type DBMS, may be in the same network. In addition to the separate DBMS and communication modules, a translator software module exists to translate between the incompatible DBMSs.

This thesis used the approach for the heterogeneous architecture and the design Capt John G. Boeckman developed in his thesis (Boeckman, 1984:20-56). It also incorporated some of the data items for a data dictionary that 2Lt Anthony J. Jones specified in his design of a global language for a DDBMS (Jones, 1984:149-153). With a global database model, all query and update functions passed over the network were written in one common language. When queries arrived at a host computer or when the results returned to the network, software modules translated the request from the global language to the host DBMS language and vice versa. As summarized in the following section, Boeckman's and Jones' theses and other studies outlined specific elements included in such a DDBMS.

Summary of Current Knowledge

Ceri and Pelagatti (Ceri, 1984:13-14) explained the basic software components of a DDBMS as the (Figure 2):

1) Database management component (DB),
2) Data communication component (DC),
3) Data dictionary component (DD), and
4) Distributed database component (DDB).

They explained the services of these components included:

1) Remote database access by an application program; this feature is the most important

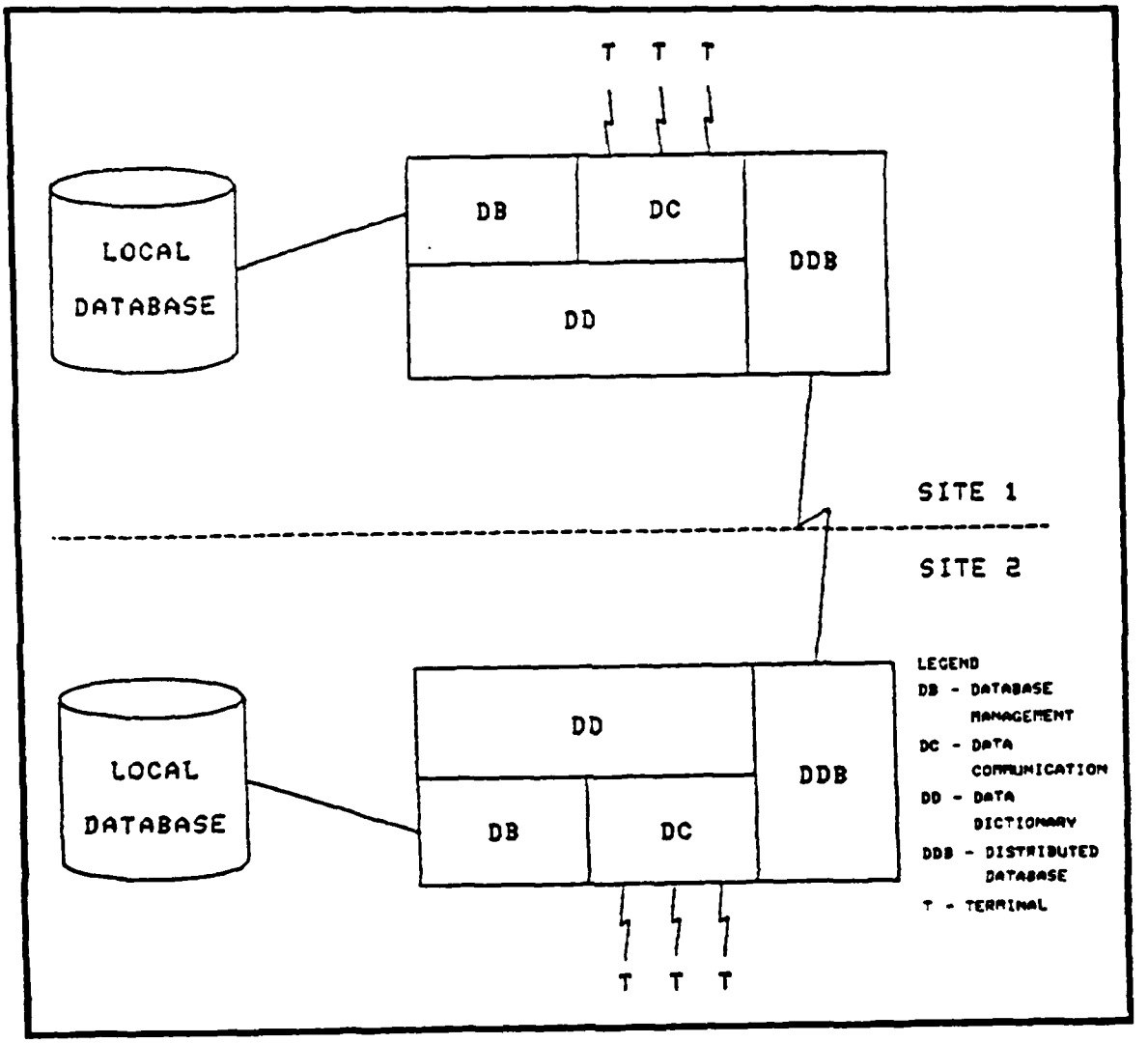


Figure 2. Software Components of a DDBMS (Ceri, 1984:13)

one and is provided by all systems which have a distributed database component.

2) Some degree of distribution transparency; this feature is supported to a different extent by different systems, because there is a strong trade-off between distribution transparency and performance.

3) Support for database administration and control; this feature includes tools for monitoring the database, gathering information about database utilization, and providing a global view of data files existing at the various sites.

4) Some support for concurrency control and recovery of distributed transactions.

Imker's high level design of a DDBMS was similar to the outline just described (Imker, 1982:63-79). He divided the DDBMS software into three parts: the Network Access Process (NAP), Network Database Management System and Network Data Directory.

The NAP is the data communications component which links the computers in a network. In the AFIT Digital Engineering Laboratory, a network operating system (NETOS) fulfills this role. NETOS follows the seven-layer protocol described in the Reference Model of Open Systems Interconnections (OSI) which the International Standards Organization (ISO) developed. Each layer has specific functions (Tanenbaum, 1981:453-487) and only communicates with its adjacent layers by calling loosely coupled modules.

The DDBMS application software performs the functions of the Application Layer, ISO layer seven. The DDBMS software also does the functions of the Presentation Layer, layer six.

That is, it prepares the DDBMS inputs in standard NETOS formats. Finally, the DDBMS interfaces with NETOS at layer six.

Another component of Imker's design was a network database management system (NDBMS). This is equivalent to the distributed database component Ceri and Pelagatti described. It is the main software module at each site and interfaces between the local DBMS and the DDBMS. This component provides links with the user, the local DBMS, the directory of data stored in the local DBMS, and the network.

The next part of Imker's design was the Network Data Directory, which this thesis designed in detail and implemented. According to Allen (Allen, 1982:246), this software component has two functions:

1) Provide the relationships between the application programs and system data usage.

2) Achieve data independence--the users can get data without knowing its location or characteristics.

Durrell (Durrell, 1983:12-19) points out several benefits of a data directory. It can be used as a communication tool, as a safequard against data redundancy, and as a glossary of definitions. Also, it helps in system development, maintenance and documentation.

Allen also listed the following components of a data dictionary/directory (D/D) system (Allen, 1982:268):

1) Database used in D/D to describe metadata, i.e.

data about data entities, processes, and users.

2) Retrieval and analysis capabilities to help develop application programs.

3) Management tools for security, validity, recoverability, integrity and shared access of the D/D.

4) Function interfaces to permit other software to access the D/D and to convert metadata to the format required by the D/D.

There are several ways to organize a directory system. Imker, in his design, used the first three of the following types of directories. A centralized directory, which Imker called a centralized network data directory (CNDD), is stored only on one system. It has a conceptual view of the data entities in all the DBMSs. An extended directory, called an extended centralized network data directory (ECNDD) in Imker's design, is a small version of the CNDD. That is, whenever a site requests the location of data from the CNDD, the local site copies the information into its own ECNDD so it does not have to ask the CNDD for the location again. Imker called the third type of directory a local network data directory (LNDD). This is a directory of only the data in the site's DBMS. The last kind, called a distributed directory, was not included in Imker's design. In this system, each computer has a complete copy of the CNDD.

Chu did cost performance tradeoffs between these different types of data directories (Chu, 1976:577-587). He

suggested a different type of directory based on the ratio between the number of directory updates to the number of directory queries. He preferred the distributed directory if the ratio was less than 10%. If the ratio was between 10% and 50%, the extended directory was best. Finally, he preferred the local directory if the ratio was greater than 50%.

In conclusion, although Boeckman did not use Imker's complete high-level design, he did incorporate in his detailed design all three directories Imker proposed. Because Boeckman did not implement the data directory system, this thesis designed and implemented this software for the DDBMS in the AFIT Digital Engineering Laboratory (DEL).

Problem

This project further refined Boeckman's DDBMS design of the central site's functions. The objectives of this research were to:

a) Design, implement, and initialize the CNDD.

b) Implement the software to request data locations stored in the CNDD, and

c) Implement the processing to retrieve data locations stored in the CNDD.

Scope

The detailed design of this project complied with Boeckman's overall system design requirements so the central site software will integrate with other parts that others

will implement later. This thesis only designed and implemented those modules necessary to service the requests for data locations.

Since there was no global translator implemented yet which would allow heterogeneous (incompatible) DBMSs to communicate, this project's implementation used the translators Boeckman used. However, this system was compatible with Jones' requirements for a DDBMS global language (Jones, 1984:149-153). By doing this, the translator modules used in this thesis can be replaced by global language modules in a follow-on thesis after the global language is implemented. Also, because these translators cannot make updates to the databases, this implementation did not maintain files to store pending updates to data for inactive sites. As a result, the tests only made queries for information stored in the DBMSs and therefore, did not update the data.

Finally, this thesis did not implement other functions Boeckman included in his design for the central system. For example, this thesis did not design nor develop the modules required to automatically reconfigure the CNDD in case it was destroyed. Nor did the thesis plan to develop a commercial-type data dictionary. This would include database management tools like statistical reports.

Assumptions

One of the assumptions of this thesis was that the design of the DDBMS that Boeckman developed was acceptable to

the user. This thesis then provided more design details on the directory system without changing Boeckman's basic design of using three types of directories.

This thesis also assumed the partial DDBMS Boeckman implemented worked correctly. That is, a person should have been able to make a query from one terminal and receive a result from either of the databases in the system. This also implied the network communication software worked correctly.

Approach

This project followed the life cycle procedures advocated in software engineering to solve the problem, namely:

a) Requirements analysis,

b) Detailed design,

c) Implementation, and

d) Integration testing.

During the requirements analysis, the first step involved learning how to operate the system Boeckman implemented and analyzing his design. At the same time, the analysis phase included a background literature search of the general data contents of a CNDD, alternative ways to build a CNDD, and Jones' requirements for the global language. Also, this analysis described the general functions of the software modules needed for this project. Finally, structured analysis and design technique (SADT) diagrams graphically showed all of these requirements (See Appendix G).

Once the requirements were defined, the detailed design described the data structures (formats) of the CNDD, the data passed to and from and software modules, and the algorithms (procedures) required to do each of the modules' functions. Structure charts graphically showed this detailed design (See Appendix I). They also identified information passed over the network which should be monitored during the testing phase. These requirements for monitoring messages were passed to Capt Janice Rowe, who was concurrently working on a network performance monitor, so the monitor can also test this project (Rowe, 1985). Finally, verification testing checked that this design fulfilled the requirements defined in the previous phase and those Boeckman defined for the overall DDBMS.

Next, the implementation phase produced software modules in the programming language C. The CNDD used abstract data types so that its implementation method did not affect the way high-level modules requested information in the CNDD. For example, a call for an abstract data entity would not change whether the CNDD was implemented in C data structures or in a DBMS. Only the lowest level module that communicated directly with the CNDD would change if the CNDD was implemented a different way. Coding adhered to the detailed design and executed on one of the computers connected to the DDBMS. Boeckman's software also changed in order to link the directories into the system. During this phase a test plan

described the procedures to check each module separately. Tests verified the accuracy of passing the locations of data stored in the CNDD to sites.

In the last phase, the integration tests also used the test plan. This phase integrated all modules to perform a full system test. The tests verified if all software modules of the central site worked together correctly and complied with system requirements.

Overview of the Thesis

The thesis format follows the approach just explained. Chapter II describes the requirements analysis of the data directory system. Chapter III then explains the detailed design of the directory used in the DDBMS. From this design Chapter IV describes the coding completed to implement the directory system. Chapter V presents the testing methods used to integrate all the modules and check the effectiveness of the system's requirements analysis, design, and coding phases. Finally, Chapter VI summarizes the results of the thesis and presents recommendations for follow-on research.

II. ANALYSIS OF REQUIREMENTS

Introduction

The requirements for this thesis were based on those already established in Boeckman's overall design of a DDBMS (Boeckman, 1984:20-36 & Vol II) and in Jones' design of a global language for a DDBMS (Jones, 1984:149-153). Since this thesis covered the portion of a DDBMS which dealt with the data directory system, this chapter only describes the requirements for implementing the directory system. The Structured Analysis and Design Technique (SADT) (Peters, 1981:62-64) was used to describe the requirements. Appendix G shows the SADT diagrams Boeckman wrote to describe the functions of the directories and the additional SADT diagrams developed in this thesis to further break down some of the functions. The next section explains the general software functional requirements of the data directory system and the following sections describe each general area in more detail.

General Functional Requirements

The central site which controls the directory system has the following functions (Boeckman, 1984:20-21):

1. Initialize the DDBMS
2. Update the CNDD with changes made in the LNDD
3. Send updates to Extended Centralized Network Data Directories (ECNDD) which contain copies of data changed in the CNDD

4. Service the Centralized Network Data Directory (CNDD) site requests

5. Reconfigure the DDBMS

Initialization of the DDBMS occurs when the system starts up. Different procedures occur depending on whether the site is the central site or not. If it is the central site, the software initializes the CNDD, queries the other sites, evaluates their responses, and sends a startup message to all the sites participating in the DDBMS. If the site is not the central site, software initializes the site's database and responds to the central site's query.

In order to send queries to the correct database, the three types of network data directories must be kept up-to-date. First, the CNDD, which is only kept at the central site, has a complete view of all the data in the DDBMS. It stores the locations of all data entities. Second, the LNDD at each site maintains information on only the data in its DBMS. Third, the ECNDD at each site keeps the locations of data that the site requested from the CNDD. This last directory makes other queries and updates to data previously retrieved from other sites faster. In conclusion, any changes to data locations in an LNDD must be reflected in the CNDD and in all ECNDDs which stored the location of the data that changed.

As a result, the central site is involved with all data queries and updates and performs three functions to service

requests. Whenever a site cannot find a data location in either its LNDD or ECNDD, it requests the location from the CNDD. Therefore, as its first function, the central site must retrieve locations from the CNDD. Secondly, if data moves from one DBMS to another, the central site must update the CNDD and notify the affected ECNDDs. Finally, the central site must manage a pending update file for each site that is inactive. This file stores all changes users make to data stored in sites that are temporarily disconnected from the DDBMS.

Not only does data change, but also the system configuration changes. In this case the central site must control the reconfiguration of either adding a site or deleting a site. If a new site is added, the new site must notify the central site and all others. The central site, in turn, sends all data updates to the new site if the central site had a pending update file with information for the new site. Then the central site notifies all sites that a new site is added to the DDBMS. On the other hand, if a site is deleted, the site must notify all other sites. At other times, if there is a malfunction and a site abnormally disconnects from the DDBMS, another site must notify all other sites of the site deletion. Also, the central site begins a pending update file for the deleted site.

Unlike other sites, if the central site is deleted, there are additional steps in the reconfiguration process.

If the central site is deleted, the central database administrator in charge of the DDBMS must choose another site as the new central site. Then the new central site copies the CNDD and pending update files from the old central site. However, if the central site malfunctions before it can copy its data to another designated central site, the new central site must read each site's LNDD to recreate the CNDD.

Detailed Requirements

Figure 3 (Boeckman, 1984:Vol II) shows the SADT design for initializing the DDBMS. It consists of initializing the central site and other sites. Other SADT diagrams contained in Appendix G of this thesis show the design for reconfiguring the DDBMS, updating and maintaining the ECNDD and LNDD, and servicing requests for the CNDD and other sites. The following sections will explain these requirements in more detail.

Initialize DDBMS. These software modules prepare the DDBMS for execution (Appendix G, SADT #C4). After the central site is chosen, the software at that site activates the CNDD and asks the operator which sites will be in the DDMBS. From that information, the central site sends query messages to all sites. After the sites respond to the central site, the central site updates the status information and issues a ready command to the sites to begin execution.

Other sites initialize their status information and prepare for a contact message from the central site (Appendix

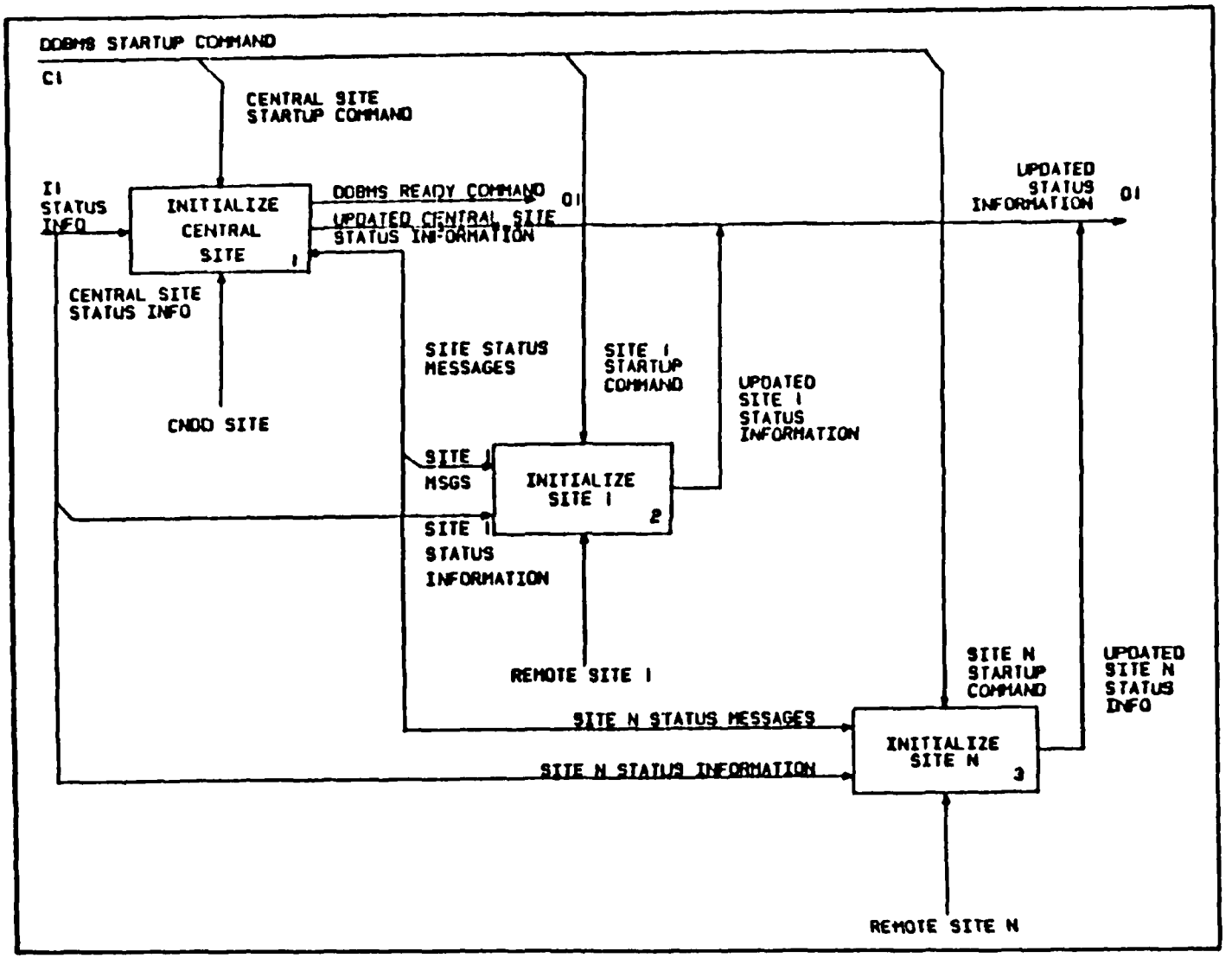


Figure 3. Initialize DDBMS (Boeckman, 1984:Vol II)

G, SADT #C5). When they receive it, they update their status information with the CNDD's location and return the information that the central site requested. Finally, when they receive the startup message, they begin executing the DDBMS.

Reconfigure DDBMS. The DDBMS may reconfigure or change its configuration whenever a site is added to or deleted from the network (Appendix G, SADT #C6). Either an operator can enter a command to reconfigure the system, or else malfunctions will cause the system to automatically reconfigure.

When an operator wants to add a non-CNDD site (one that does not contain the CNDD) (Appendix G, SADT #C7), he first sends a contact message to the central site. The central site, in turn, updates its status information and sends an acknowledgement message to the added site. Then the added site updates its status information and its local data from information that the central site stored in a pending update file for the site. After the central site finishes sending the pending update file, it notifies all the other sites that a new site is added.

If a non-CNDD site is to be deleted (Appendix G, SADT # C7a), the operator sends a message to all sites explaining that the site is dropping off the network. Then the central site and all other active sites mark their status information accordingly. Also, the central site starts a pending update file for the deleted site.

Moving a CNDD site to another site (Appendix G, SADT #C8) requires copying the CNDD and the pending update files. When the transfer is complete, both sites adjust their status information and also notify all sites of the new CNDD location. During this reconfiguration process the central site cannot respond to any data location requests nor change its pending update files.

In case of system malfunctions (Appendix G, SADT #C9), an operator does not have to initiate the reconfiguration as in the previous three examples. Through malfunction messages the DDBMS will recover from a site crash by changing the status information and beginning a pending update file for the site. If the CNDD site fails, another site, chosen by some predetermined method, automatically recreates the CNDD by consolidating the data from all the LNDDs. When there are communication line failures, the network must reroute messages so the sites can communicate between each other. Finally, after the central site makes all the necessary changes to its status tables, it sends to all the sites information on the new DDBMS configuration.

Updating and Maintaining the ECNDD and LNDD. Two of the functions of executing the DDBMS at the sites is to update the ECNDD from CNDD updates and to update and maintain the LNDD (Appendix G, SADT #C13). When there are updates to the CNDD, the central site must determine what sites had requested the locations of the data that changed. Then the central

site sends changes to these sites so they can change their ECNDD. After a site receives the ECNDD update (Appendix G, SADT #C13a), it must make the changes to its ECNDD. Finally, after making the changes, it sends an ECNDD update acknowledgement message to the central site.

The second function to update and maintain the LNDD is necessary to keep the LNDD current with the local DBMS (Appendix G, SADT #C13b). When external user inputs change the database which require changes in the LNDD, the site must notify the CNDD of the changes. However, the site software does not change the LNDD until it receives an acknowledgement message that the CNDD made the changes.

Service Request at Sites Other than the Central Site. Another site function while executing the DDBMS is to service requests from this site and other sites in the DDBMS. If the query originates at the local site, it is called a local query. Otherwise, if the query at the site comes from another site, it is a remote query.

To service local queries (Appendix G, SADT #C16), software first determines the query type by searching for the data's location in the site's LNDD. If the site has all of the data in its host--or local--computer, it is a host query. In this case, the site can process the query without checking any other directories. On the other hand, if other computers in the DDBMS have the data, it is a network query.

To service a network local query (Appendix G, SADT #C18), the system first translates the query from the local language into a global data model language. Then software services the translated network query (Appendix G, SADT #C19). If the data location is not in the site's ECNDD (Appendix G, SADT # C19a), the site must ask the CNDD for the location. Once the data locations are known, the system continues to process the query. After the query results are completed, the site updates its ECNDD with data that was not in its ECNDD.

Besides servicing the local queries, a site may service a remote query (Appendix G, SADT #C25). For this query the site only has to check its LNDD to verify its host DBMS contains the data. If it does not have the data, the site must notify the CNDD site of the data location error in the CNDD. In this case, the site must also notify the site which originated the query.

Service Requests at Central Site. Just as with the other sites, the central site must first determine the CNDD request type (Appendix G, SADT #C28). They may be either CNDD data location requests, CNDD updates, or pending update requests.

To service CNDD data location requests (Appendix G, SADT # C28a), the CNDD site receives the site requests, which include a global relation name with its global attribute names. A global name is a common name used for possibly several alternate names used in different DBMSs. Then the CNDD determines the data locations. The CNDD site will send all

the locations of the data if it is redundant, horizontally partitioned or vertically partitioned. Redundant relations are those that have identical structures (i.e. they have the same attributes) and duplicate data. According to Ullman (Ullman, 1982:411), if relations are horizontally split, two or more relations contain the same attributes but the relations contain different information. On the other hand, Ullman states a vertically partitioned relation has attributes which are physically located at different sites. For example, a global relation may contain three attributes A, B, and C. One DBMS may contain the relation with attributes A and B, whereas another may contain the relation with attributes B and C.

To service CNDD updates due to LNDD updates (Appendix G, SADT #C29a), the CNDD site receives the CNDD updates from another site and matches the received data against the data in the CNDD. Next it updates the CNDD and sends an update acknowledgement message to the sending site. Then it sends updates to the ECNDDs which also have the data (Appendix G, SADT #C29). Finally, the central site receives an ECNDD update acknowledgement message from the other sites which received ECNDD updates.

The last CNDD request type is servicing pending update requests (Appendix G, SADT #C28). For this request, the central site adds information to the pending update file of a site that dropped from the network while the DDBMS was opera-

ting. Also, the central site sends the results of the update back to the site which originated the pending update request.

General Content of Data Directories

In his thesis Jones (Jones, 1984:149-153) presented what a data dictionary should contain when using a global relational data model. It included information about the databases in the system, what relations were stored in each database, the attributes of each relation and other information needed to map--or translate--from the global relational language to a local database definition language.

Since no global relational language has been implemented yet, this thesis did not include all the data requirements Jones presented. Instead, this thesis only used those items Jones described which were necessary to locate an entity within a database. Other information needed for completely mapping a global to local data definition language, and vice versa, may be added as a follow-on effort to this thesis.

In the list of items in the directories "identification" and "name" are used several times. An identification code is a unique number or unique character string, which is used as a key in several of the relations in the CNDD implementation. In contrast, a name is a descriptive, nonunique character string used in one of the databases. Since the same name could be used for different items, the unique identification code, rather than the name, was used in several places to establish links between different CNDD relations.

Based on Jones' research, the following information was included in the CNDD and ECNDD:

a. Site identification of source (identifies the network address of the site)

b. Host computer (e.g. UNIX VAX)

c. DB name (e.g. AFIT, Demo, etc.)

d. Global relation name ("Global" name is a common name for possibly several local relations with different names stored in separate databases. A global relation identification was not needed because the global relation name must uniquely identify the relation.)

e. Relation replication code (specifies whether data is duplicated in several databases and how the data is partitioned)

f. Global attribute identification

g. Global attribute name

h. Local relation identification ("Local" relation is a relation stored at a local database. If the local DBMS was a network or hierarchical type DBMS, the entity was translated to a relational type before storing it in the directory. In a concurrent effort with this thesis, Capt Kevin Mahoney (Mahoney 1985) stored the mapping information needed for this translation elsewhere.)

i. Local relation name

j. Local attribute identification

k. Local attribute name

In addition to Jones' requirements, this thesis added the following items which were necessary to implement the directory system:

a. Access code (prevents CNDD from releasing data that is being updated)

b. DBMS name (e.g. DBTG, INGRES, dBASE II, Total)

c. DBMS type (e.g. hierarchical, relational or network)

d. Local relation index code (specifies whether the relation is indexed on a particular attribute)

As for the LNDDs, they do not need to store their own site identification and site name. However, besides this information listed, other information needed to map data definitions from one type of DBMS to another should be stored in the LNDD. The LNDD should store it because the processing should not have to convert from the global relational data descriptions to that used in a host database until just before sending a query to the host database. Therefore, when a site receives a query to send to its host DBMS, the processing should extract the mapping information from the site's LNDD to make the data definition translations. Since it was not in the scope of this thesis to design and implement the global language translator, this thesis did not list all the mapping information.

Summary

This requirements analysis discussed the four general functions of the data directory system and graphically decomposed the requirements using SADT diagrams. The software modules consist of those to initialize the DDBMS, reconfigure the DDBMS, update and maintain the ECNDD and LNDD, and service CNDD site requests. Also, the analysis described the general data elements of the data directories. The following chapter explains the detailed design for these requirements.

III. DETAILED DESIGN

Introduction

This chapter adds to the detailed design Boeckman presented (Boeckman, 1984:37-56), where necessary, to be able to implement the centralized data directory system. In particular, the following sections will describe the software processes to service the CNDD site requests and update the LNDDs, two of the central site's functions. The other central site functions of initializing the DDBMS, updating the ECNDDs and reconfiguring the DDBMS will not be discussed because the detailed design did not change from what Boeckman presented.

The detailed design used structure charts and process and parameter data dictionary entries that are located in Appendices I and J. The structure charts described the hierarchy of software modules and the data passed between modules. The process data dictionary entries explained the purpose of the modules, the relationships between the modules, and the modules' input and output data. The parameter data dictionary entries described the parameters' use and characteristics of these input and output data.

Further Decomposition of Requirements

As the requirements were further decomposed and implementation decisions made, there were limitations placed on the requirements. For instance, there were restrictions on the form of the CNDD data definitions and the ability to move the CNDD from site to site. This implementation first re-

stricted the CNDD to use relational data definitions. In other words, a data definition in a network DBMS had to be converted via some algorithm to a relational form to be stored in the CNDD. For example, Jones described how to map from the network and hierarchical data definition languages to a relational data definition, and vice versa (Jones, 1984: 115-137). The main reason the CNDD listed relations and attributes was because the queries were written in the Roth relational data manipulation language (Roth, 1979:122-124) developed at AFIT.

Figure 4 shows the CNDD has a global view of all the DDBMS data stored in a relational data definition language. That is, the data schema is described as attributes within relations. Since the LNDDs must be used to build a new CNDD when the original CNDD site fails, they also must describe the schema in terms of a relational data definition language. However, the LNDDs must contain extra information not needed in the CNDD in order to map--or translate--from the relational data definitions to the actual data definitions used in the host DBMS, which may be a network, hierarchical or relational type of DBMS. Appendix A shows the definitions of the data in the CNDD, and Appendix D shows those in the LNDD. A separate description of the ECNDD was not included because it contains the same type of information as the CNDD.

The requirement to be able to move the CNDD from one site to another was also restricted because of implementation

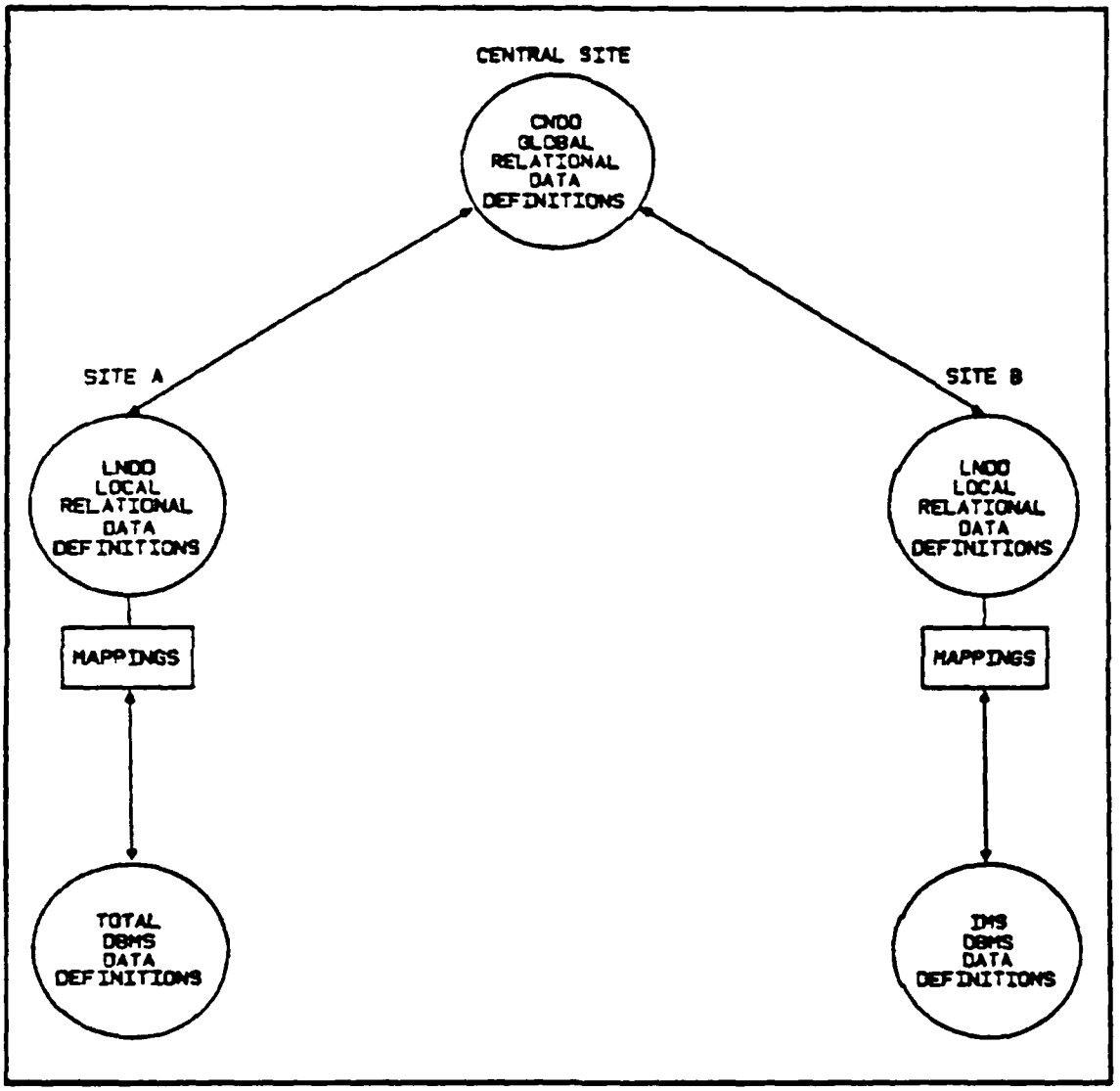


Figure 4. Data Directories Data Definitions

decisions. The general DDBMS design specifies that the CNDD should be able to move from one site to another in case of failure at the central site. However, the CNDD was implemented on a host computer DBMS because the DBMS already provided data manipulation routines. Therefore, the DDBMS cannot move the CNDD to a secondary site unless the lowest level modules which interact with the CNDD at the secondary site are also implemented. In other words, if the secondary site stores the CNDD in another type of DBMS, the modules which extract data from the CNDD must interface with the specific host DBMS. As a result, all sites are designed to have the same software for the upper level modules necessary to act as the CNDD site, but the code of the lower level modules will differ based on how the CNDD is implemented at the particular site.

This restriction would not be necesary, though, if the CNDD were implemented the same way at all sites. Each site could have the same software to process CNDD requests and therefore, could be interchangable. For example, every site could define the same data structures for the CNDD in the common software modules executed at all sites. Then the routines to manipulate the CNDD would be the same at all the sites. However, this method requires that the developer design and code all the data manipulation routines already found in a DBMS. For example, a DBMS has software to define data characteristics, update and access the data and maintain

data integrity. Therefore, it is faster to implement a CNDD by using a DBMS.

Structure Chart Design

The following sections in this chapter describe the structure chart design and data passed between those modules which support the central site's functions. According to the system requirements, all the software to implement this design should be on all sites in the DDBMS. However, if the site is not the CNDD site, the modules to process the CNDD site requests will be turned off. The next sections describe the detailed design in this thesis that was expanded beyond Boeckman's design to support the following functions:

1) Service Centralized Network Data Directory (CNDD) site requests

2) Update the Local Network Data Directories (LNDD)

Service CNDD Site Requests

The structure chart in Figure 5 shows three different kinds of requests the CNDD site processes: data location requests, CNDD updates and pending update requests. Since the central site software is part of every site's software, the site first checks if it is the operating CNDD site. If it is, it continues to process one of the three kinds of requests. Otherwise, the site sends an error message back to the requesting site explaining it cannot process the request.

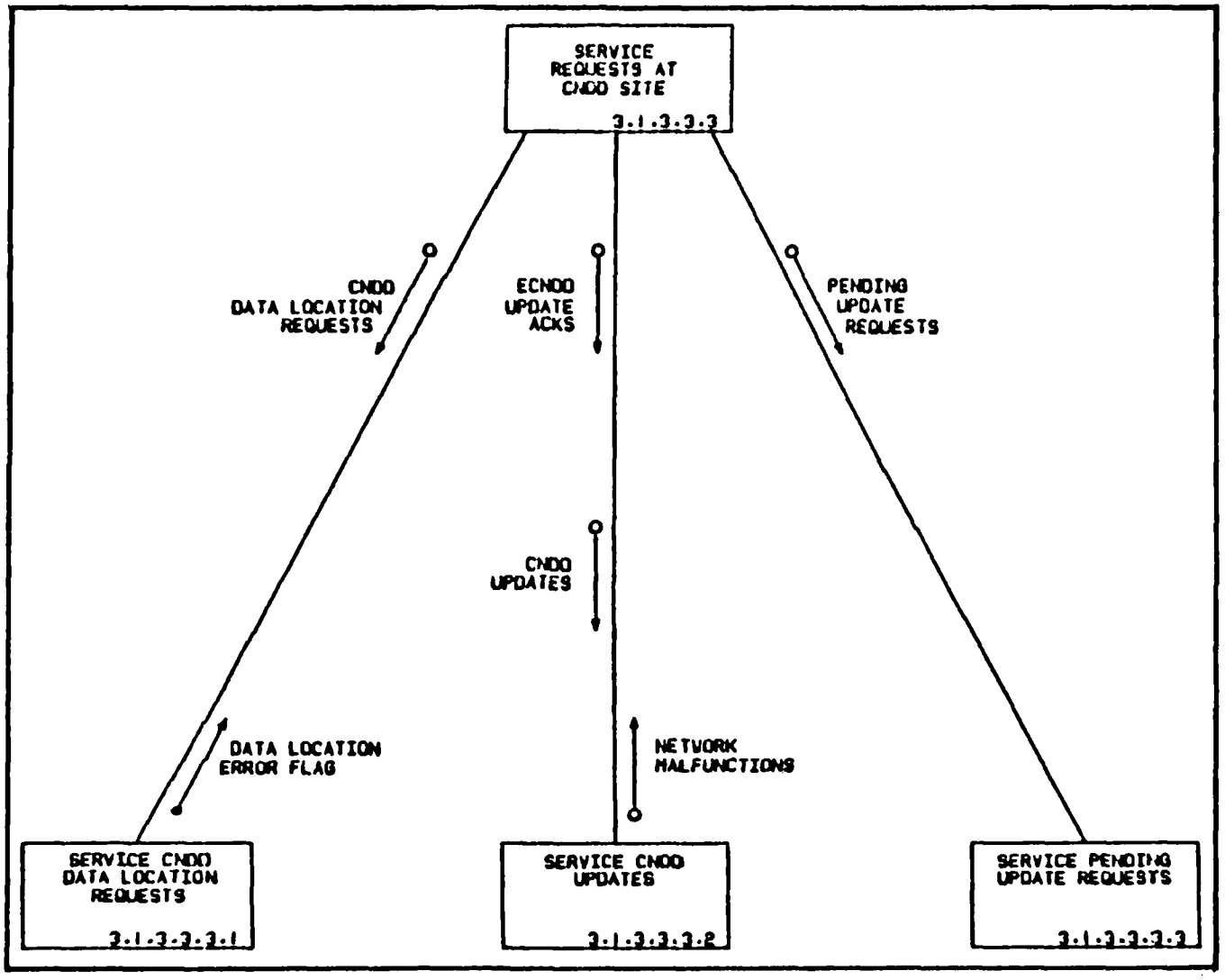


Figure 5. Service Requests at CNDD Site

The following section explains in detail how the CNDD site services data location requests. After this explanation the chapter explains the conceptual procedures for updating the CNDD. This is not as detailed as that explained for servicing data location requests because it was not the intent of this thesis to implement CNDD updates. Also, this chapter does not discuss the design of processing pending update requests since it was out of the scope of the thesis.

Data Location Requests. For data location requests, the central site first verifies whether the CNDD Data Location Request message (see Appendix E) contains the correct password in order to access the CNDD. There is only one password for general access to the CNDD. The CNDD itself does not check whether the user has access privileges to a specific database or to data within a database. The individual DBMS has the responsibility to control access to its database when it receives a query message, which also contains a password.

After checking the password, the software then extracts information from the request message in order to build a standard header for the results message, which will contain all of the data location information retrieved from the CNDD. The information extracted from the request message includes the requesting site's identification code and the query identification code. The CNDD site uses the requesting site's identification as the destination for the results message it will send back at the end of the processing. The query

identification code has another purpose. The network optimizing software assigns a unique query identification code to each user's query. Then the optimizer divides the query into subqueries to send to different sites to get results for a user's original query. Each subquery will carry the same query identification code. In this way the DDBMS optimizing modules can combine all the results from several host DBMSs into a final response.

Since the user's query is written in a relational data manipulation language, the query includes names of relations and attributes. From the user's viewpoint these relation and attribute names are global names. In other words, they are names used at the highest conceptual level with which the user is familiar. Hence, the goal of the CNDD data location software is to specify which DBMS in the network contains local relations which are components of the global relation.

The local relation and local attribute names are those names used in a specific host database. The local names may be different from the global names or the same as the global names. Even if the same, though, they may not match conceptually with the global data. In other words, a central database administrator has to decide which local relations contain data that are defined as part of each global relation. Then he includes these mappings in the CNDD.

The CNDD software was designed so that the modules retrieve the data locations in two different ways. It can

search for either the locations of specific global attributes within a global relation or the locations of all global attributes defined to be part of a global relation. Therefore, the CNDD Data Location Request message includes a request type designator before each global relation name. Type 1 informs the CNDD to extract the locations of all the global attributes within the specified global relation. Type 2, on the other hand, signifies to get the locations of only the global attributes listed after the global relation name. For each relation listed in the request message, the CNDD software finds out what the request type is and the name of the relation.

Because of the overhead required in the message header information, this design allowed several data location requests to be combined into one message. If there were only one request per message for a global relation's data locations, each message would have more header information than the name of the relation. Therefore, it was more efficient to combine the requests.

As a result, Figure 6 shows four high-level steps of servicing a CNDD data location request. First a module gets the request type and a global relation name from the request message. This step was added to Boeckman's design because of the decision to combine several requests into one message. Next, the CNDD processing extracts the data locations of one relation at a time. Then it reformats the information re-

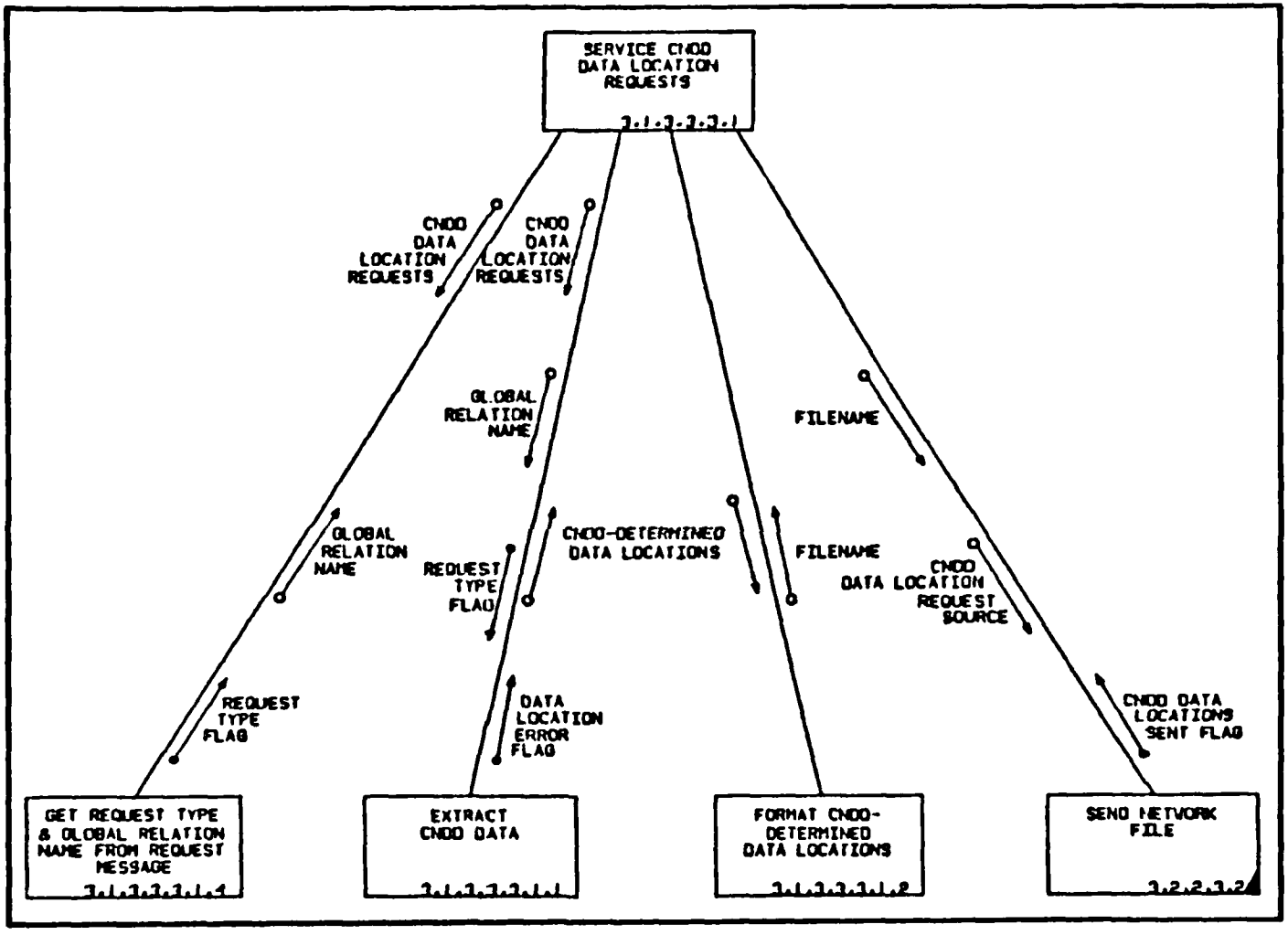


Figure 6. Service CNDD Data Location Requests

turned from the CNDD into the CNDD Data Location Results message (see Appendix E). These first three steps continue until the CNDD finds the locations of all the relations and attributes in the request message. Finally, the CNDD site sends the results message to the requesting site.

In order to extract the data locations requested, a software module first checks if the CNDD contains the global relation in its directory, as depicted in Figure 7. If it does not exist in the CNDD, the software notes it in the results message and then continues the processing for the next relation in the request message. If the CNDD does contain information on the relation, it next checks whether access to the data locations is locked or not. The CNDD prevents access to the information for a global relation while the CNDD is updating any data on the relation. This prevents the CNDD from sending back inaccurate information to the requesting site.

Finally, the lowest level modules retrieve the data locations of the global relations depending on the type of request. For example, a type 1 data location request would probably be used for a SELECT relational query. In relational algebra, relations are represented as tables (with rows and columns) of data. As C. J. Date explained, "The SELECT operator constructs a new table by taking a horizontal subset of an existing table, that is, all rows of an existing table that satisfy some condition" (Date, 1982:75). Since a SELECT

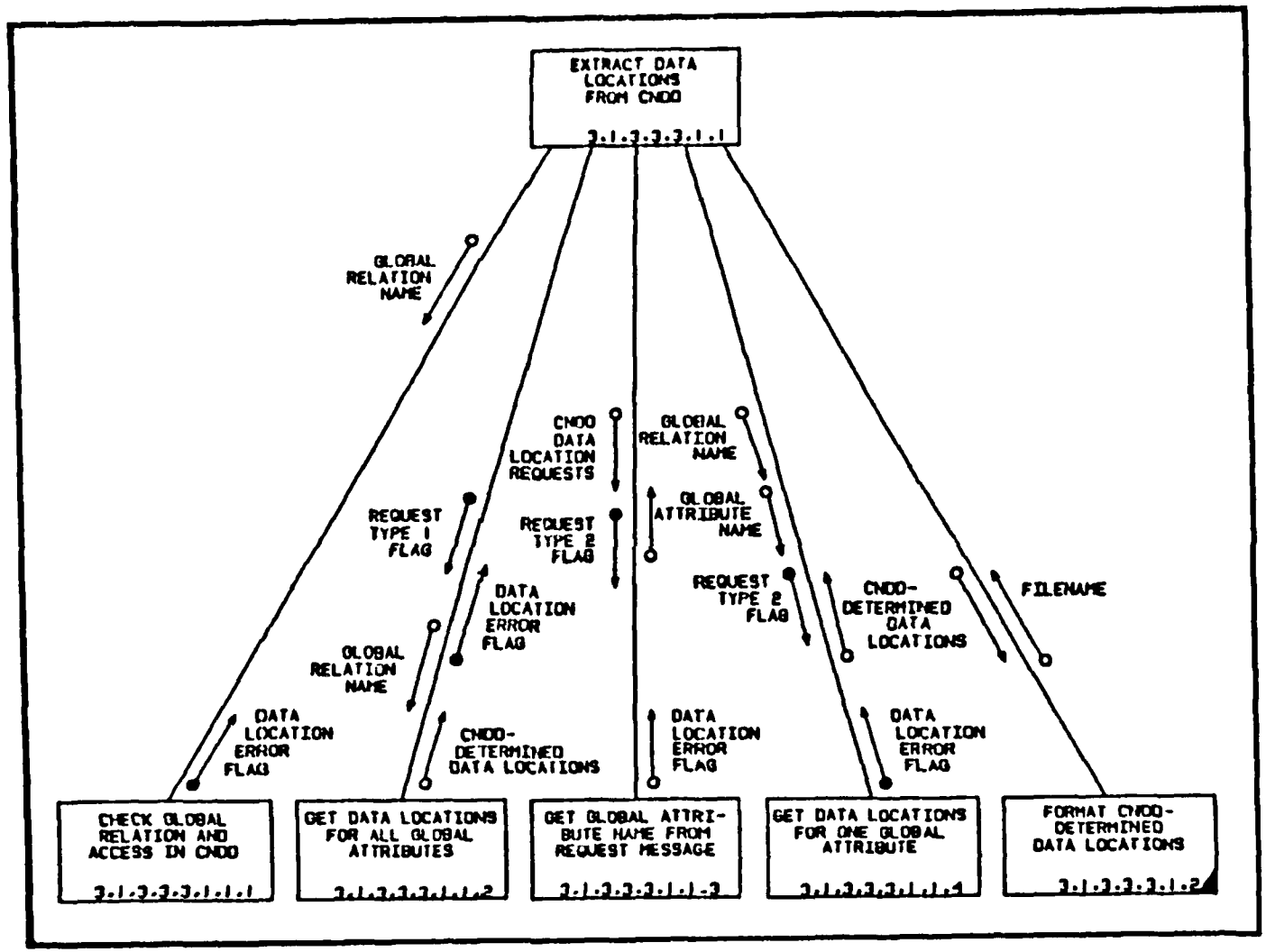


Figure 7. Extract Data Locations from CNDD

operation returns entire rows of a table--or tuples--which include all attributes within the global relation, the DDBMS optimizing software must know the locations of all the relation's attributes. In contrast, a PROJECT relational query would probably require a type 2 data location request. The PROJECT operator in relational algebra "forms a vertical subset of an existing table by extracting specified columns" (Date, 1982:75). Therefore, since only specified columns--or attributes--are returned, the optimizing modules need the locations of only some of the attributes.

In the case of the type 1 request, the CNDD software retrieves the locations of all global attributes within the specified relation. Before retrieving any data from the CNDD, a software module checks if there are any global attributes stored in the directory that are associated with the global relation. There should always be attributes defined for each relation in the CNDD unless the directory was not built correctly. If there are no attributes defined in the CNDD, the software notes it in the results message and continues to process the next relation. In the normal case when there are attributes in the directory, the software retrieves the data locations of all the attributes at one time. All of the information is compiled into one file and then reformatted into the results message.

In contrast, the type 2 request finds the locations of each attribute listed after the relation, one at a time.

First, a module gets a global attribute name from the request message. Then the software checks if the attribute is stored in the directory. It may not be in the CNDD if none of the sites has data for the global attribute. If it is not in the CNDD, the software processing marks it accordingly in the results message. Then it begins the cycle again to get the next attribute name in the request message. If the global attribute name is in the CNDD, the program extracts the data location information from the CNDD. After the CNDD returns the data for each global attribute, the software reformats the data to add it to the results message. This type 2 process repeats until there is another request type in the request message or else the request message ends.

When there is another request type in the message, the software reevaluates which of the above processes to follow. This entire process continues until the CNDD has searched for all the data requested. Finally, the CNDD site sends the CNDD Data Location Results message to the requesting site.

CNDD Update Requests. Another function of the CNDD is to service CNDD update requests. The following is a conceptual idea of how to process the update semi-automatically until the entire process can be automated. Part of the process must be manual because the central database administrator (DBA) responsible for controlling the update may have to make some decisions before the update can proceed. For example, if a new relation was added at a site, someone has

to decide to which global relation(s) the local relation belongs. He also has to match the local attributes within the new local relation with the global attributes within the global relation. To explain this process, Figure 8 shows the upper-level modules required to service this request.

First, when the CNDD receives an update message from a site, it locks the access to the global relation's data. This prevents the CNDD from sending to a requesting site any data location information on the global relation that is not current. Besides changing the global relation's access code, the software also changes the access codes of the specific local relation and local attribute whose data is changing. These access codes remain locked until the update is completed. Until then, the CNDD site sends a flag meaning the data is being updated, rather than the data location information, to each site that requests information on the affected relation.

Second, the CNDD site services the updates to the CNDD sent from sites that intend to update their LNDDs. The CNDD site software displays a message on the central site's terminal explaining the changes to be made and writes the same information in a file. This allows the central DBA to review the information while the central site is off-line. After making the necessary decisions, like global relation-local relation mappings, the central DBA manually changes the CNDD when the system is off-line. He also marks that the update

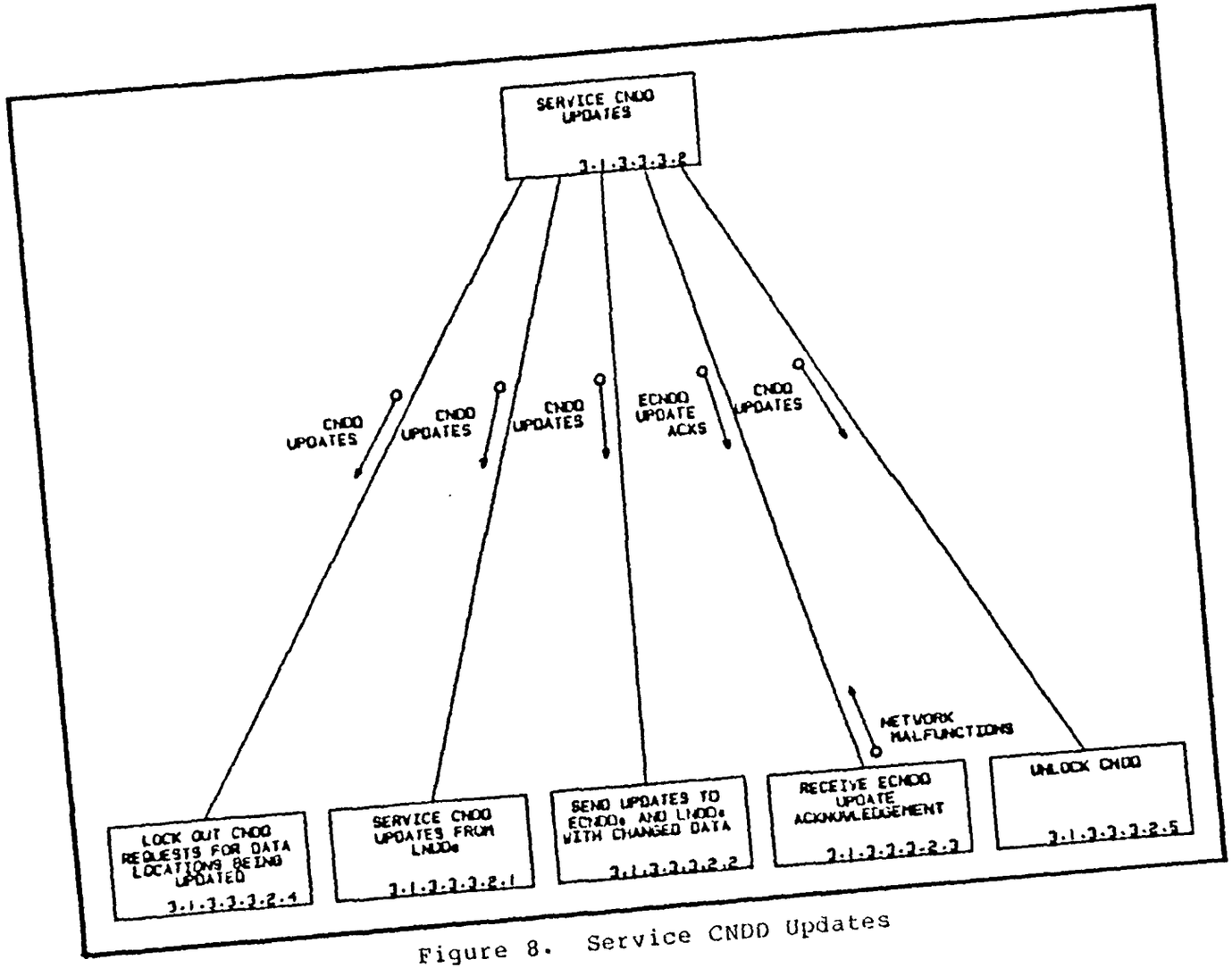


Figure 8. Service CNDD Updates

was completed in the file that contained the information on the update.

When the DDBMS comes back on-line, part of the CNDD initialization processing checks this file. If there are CNDD updates marked as completed in the file, the CNDD site finishes servicing the CNDD updates before servicing new CNDD requests. The software checks which ECNDDs and LNDDs must be changed also because they have duplicate data just changed in the CNDD. The site which originated the update is included in this list because it does not change its LNDD until after receiving an acknowledgement from the CNDD site. Also, it may need some information from the CNDD, like the global relation-local relation mapping, to store in the LNDD. The processing writes which directories and what changes are necessary in each in a file containing CNDD acknowledgements and replication data.

In the third major step to service CNDD updates shown in Figure 8, the processing sends updates to ECNDDs and LNDDs which must be changed. The software checks the file containing the CNDD acknowledgements and replication data. For each ECNDD and LNDD update in the file, it builds an ECNDD or LNDD update message and sends it to the site. When the site which originally sent the update to the CNDD receives the LNDD update message from the CNDD site, it can finally update its LNDD.

Next, the CNDD site waits for an acknowledgment message from the ECNDDs in the fourth step. After each site which received an ECNDD update message makes the directory changes, it sends an acknowledgement message to the CNDD site. When the CNDD site receives all the ECNDD acknowledgement messages it expects, it unlocks the CNDD in the fifth and final step. In other words, all access codes associated with the updated global relation, local relation and local attributes are set so any site can receive the CNDD data stored for these items.

Update the LNDDs

Since the last section just explained that the processes to update the CNDD and LNDDs are correlated, this section explains the conceptual procedures to update an LNDD. When a database administrator (DBA) wants to change data in a local DBMS, which also affects the LNDD, he must interrupt the site to notify the DDBMS of the pending LNDD update. This interrupt causes the site to receive an External LNDD Update message. This message and an LNDD Update message from the CNDD both cause the modules shown in Figure 9 to begin executing.

When the update messages arrive, the software prints a message on the site's console explaining the pending LNDD update and stores it in a file for off-line review. Because the LNDD data will be changed, the software locks the access to the affected data. Until the data is updated, the LNDD will not release any of the currently inaccurate data.

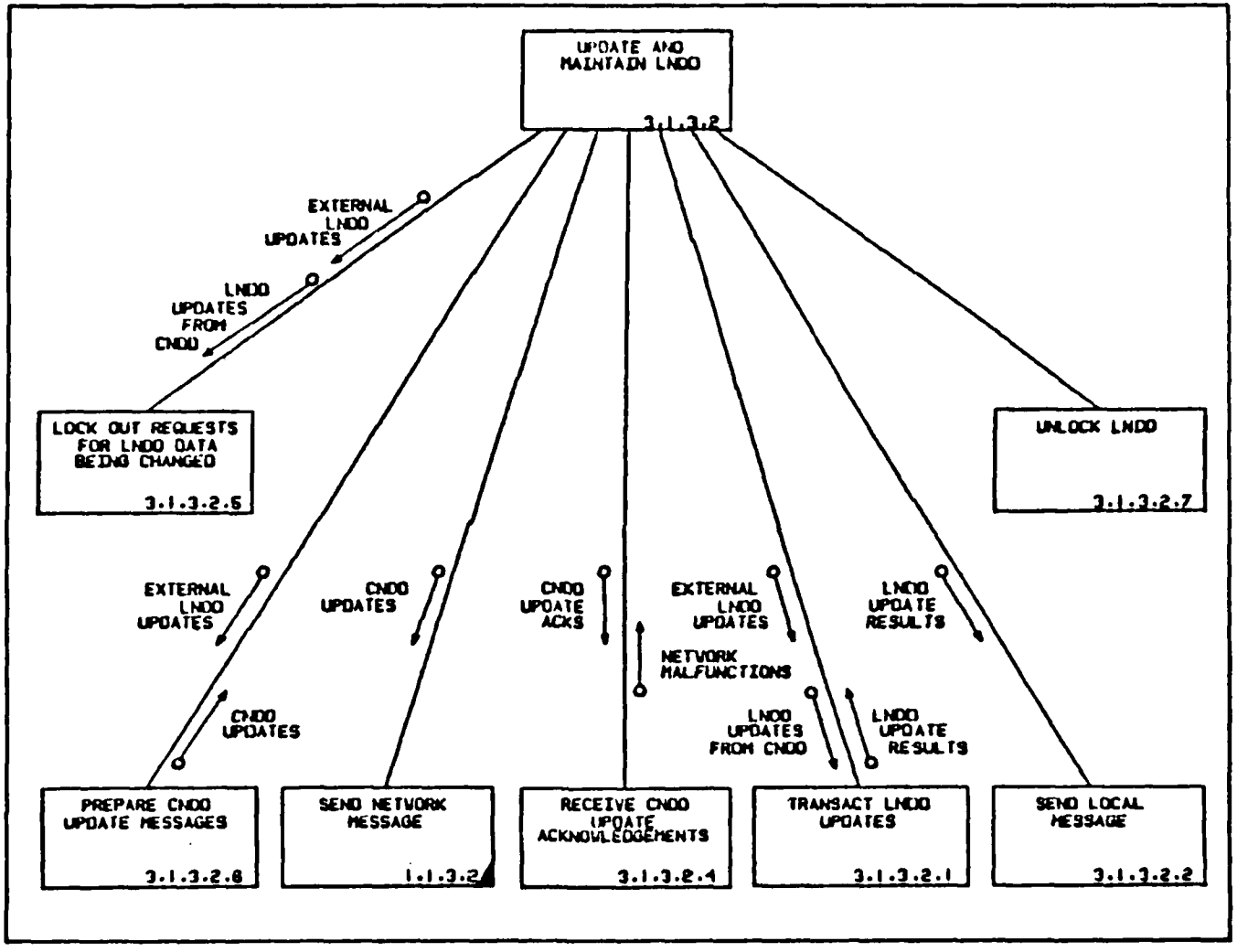


Figure 9. Update and Maintain LNDD

Next, a software module prepares a CNDD update message. This message contains local data that must be changed. For example, the DBA may want to add another field to the DBMS. If the host DBMS is not a relational DBMS, the DBA must translate the data definition of the field to a relational data definition. Perhaps the field equates to an attribute within a relation. The DBA responsible for this DBMS can only supply the local information like the local attribute name, local relation name, etc. In order to insert the global relation name and global attribute name in the LNDD, the local DBA must wait until the central DBA responsible for the entire DDBMS supplies this global information.

So the site sends the CNDD Update message to the CNDD site and then waits until it receives the CNDD Update Acknowledgement message. This message will contain the additional information the LNDD needs. When the acknowledgement message arrives, a message appears on the site console.

The next step is to transact the LNDD update. Boeckman designed this as an automatic procedure of finding the LNDD entry to be updated, changing it, and preparing a message with the update results. The system then sends the LNDD Update Results message to the host computer.

At this point the automatic procedures will probably stop. Most likely the DBA will have to take the site off-line to make the changes to the data in the host DBMS. After the changes are done, the last step is to unlock the LNDD.

This means changing the access codes of the affected global relation, local relation and local attributes in the LNDD. Finally, the LNDD is back in normal operation to determine if data is stored in the host DBMS.

Summary

This chapter described with the graphical aid of structure charts two of the central site's functions. Several sections explained these functions by detailing the process of servicing CNDD site requests and updating LNDDs. The CNDD site requests discussed included the data location requests and the CNDD update requests. In addition to explaining the software process, this chapter showed the detailed format of the messages necessary to implement these functions and the definitions of data stored in the CNDD, ECNDDs and LNDDs. The next chapter shows how the DDBMS was partially implemented based on this design.

IV. Partial Implementation

Introduction

Rather than develop another design for a partial implementation of the DDBMS as Boeckman did (Boeckman, 1984:37-56), this thesis implemented the same DDBMS detailed design described in Chapter 3 of Boeckman's thesis and this thesis. The implementation followed a top-down programming approach. In other words, the top or highest level modules shown in the structure charts were coded and tested before the rest of the system was finished. However, because of the time constraint and scope of this thesis, not all the DDBMS was implemented. Some of the modules, written as dummy stubs, can be implemented later on. Since the centralized network data directory system (CNDD) was the main thrust of this thesis, this phase of the work completed all of the processing to make a request for data from the CNDD and to get the data locations from the CNDD. Appendices I and J show the structure charts and data dictionary entries used in the implementation.

The DDBMS hardware consisted of two LSI-II microcomputers and one Z-80-based S-100 bus microcomputer. The S-100 computer executed a dBASE II DBMS, which is a relational type DBMS, and supported the CNDD.

This chapter first discusses the computer architecture used to test the DDBMS software implemented. Next, it explains how the CNDD was implemented using the dBASE II DBMS. After explaining this background, the chapter outlines the

software modules written in this implementation of the DDBMS and a summary of all the activities in this phase of the project.

Implemented Architecture

Figure 10 shows the architectural topology of the hardware used in this implementation. The DDBMS system consisted of two LSI-11 microcomputers and one Z-80-based S-100 bus microcomputer. The LSI-11 computers were identified as System L and System S in the AFIT Digital Engineering Laboratory (Hartrum, 1985:1).

One of the LSI-11 computers, System L, acted as the CNDD site in the DDBMS. Because of memory limitations, System L only contained the DDBMS software necessary to process CNDD site requests. It did not process queries or updates to the distributed databases. System L connected to an S-100 microcomputer which acted as a host computer. This S-100 executed the dBASE II DBMS to load, update and access data in the CNDD. The other LSI-11 computer, System S, was a remote DDBMS site which executed the software to handle the DDBMS queries and create data location requests for the CNDD site.

Although the hosts were nodes on LSINET, because of memory sizing problems, these LSI-11 computers were unable to contain the network operating system (NETOS) used for the LSI-11 computers to communicate between each other (Hartrum, 1985:1). The NETOS software required 34K, the DDBMS remote site software required 40K, and the CNDD site required 36K.

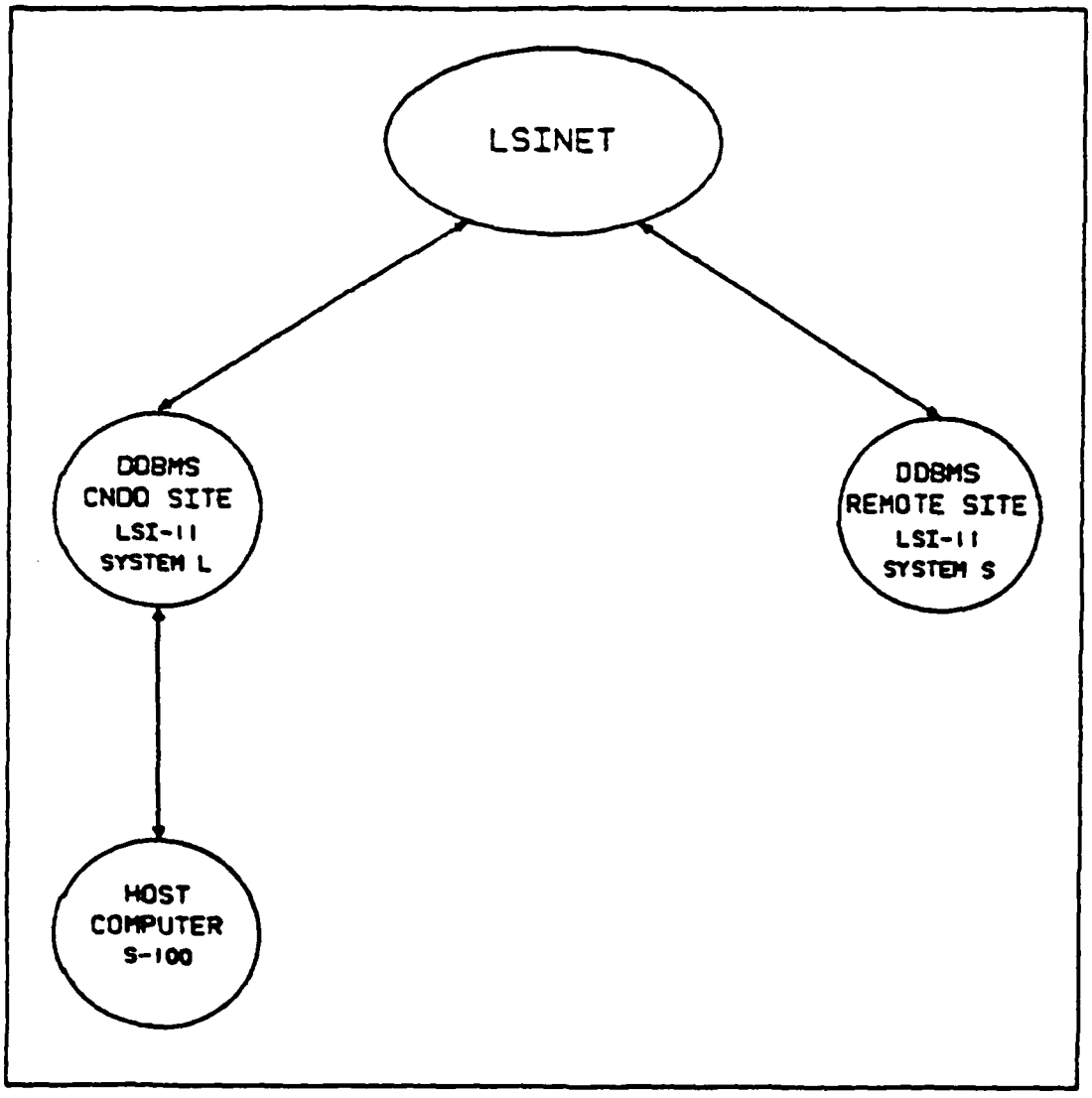


Figure 10. DDBMS Partial Implementation Architecture

Since neither the DDBMS or CNDD software is completed, the memory requirements will grow as more software is implemented. Therefore, the programs should be partitioned among the computers. For example, the LSI-11 computers could only contain the NETOS software while the host computers with larger memory capacities could run the DDBMS remote site and CNDD site software.

Implementation of CNDD

The CNDD was implemented using a DBMS just like any other database in the system. However, this site only accessed the CNDD information and did not access any of the distributed databases in the DDMBS that a user could query and update. It was decided that this site would only handle CNDD site requests because of sizing problems. In fact, the LSI-11 computer memory was not large enough to process all the CNDD site requests. Therefore, due to the memory restrictions and the scope of the thesis, only the data location requests were processed at the CNDD site.

The CNDD data shown in Appendix A was originally organized into the relations shown in Figure 11a. These original relations were all normalized to the third normal form. However, many of these relations were combined to make the CNDD processing more efficient. Figure 11b shows the final six CNDD relations formed from those in Figure 11a and loaded into a database with the dBASE II relational DBMS. In addition, Appendix B contains a User's Guide on the update procedures

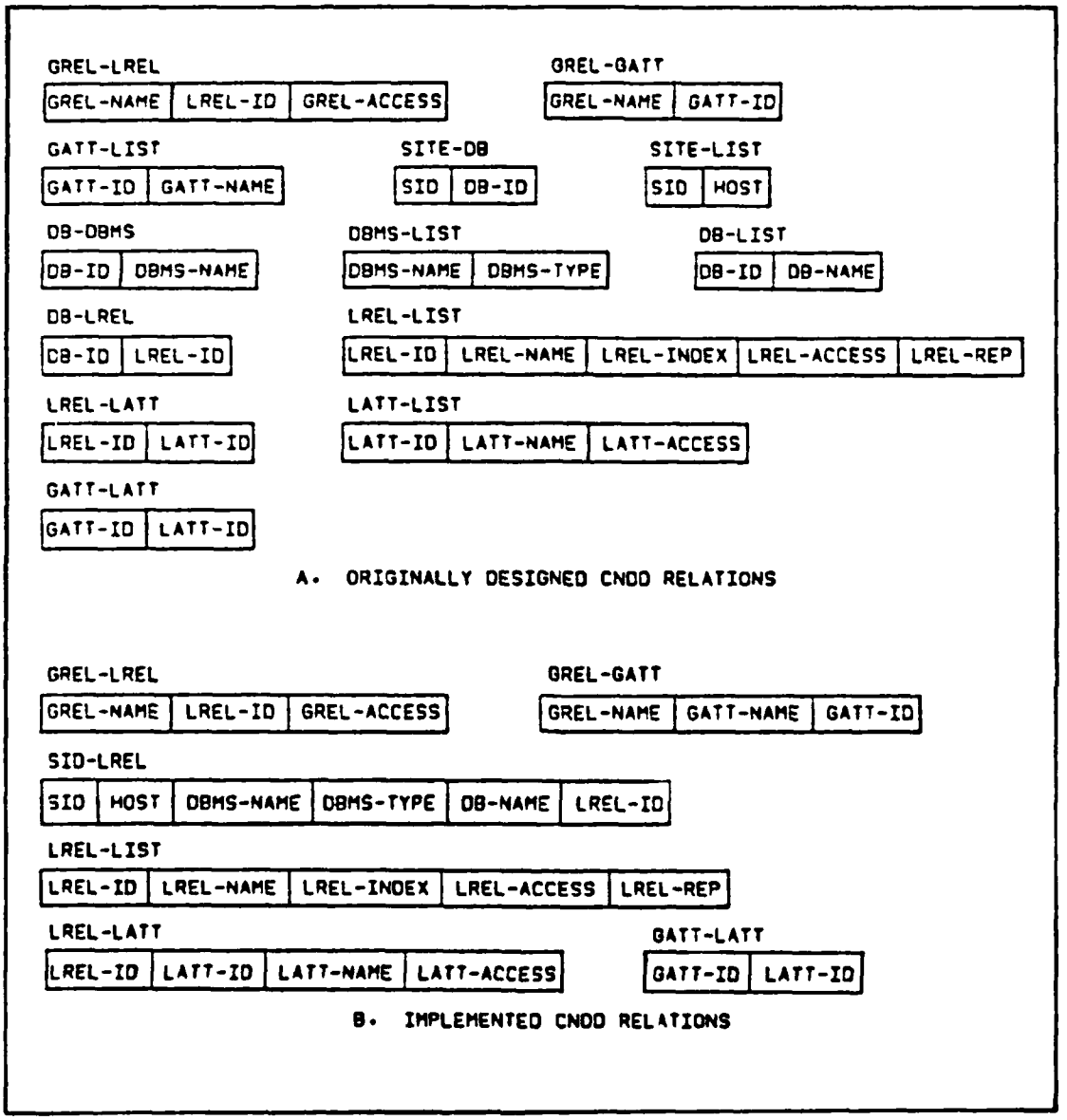


Figure 11. CNDD Relations

to maintain the CNDD, and Appendix C shows the CNDD test database constructed.

For example, the following relational algebra operations on the relations in Figure 11b retrieved the data locations of all global attributes within a global relation:

```
SELECT GREL_GATT WHERE GREL_NAME = 'RELATION'
GIVING TEMP1

JOIN TEMP1 AND GATT_LATT ON GATT_ID GIVING TEMP2

JOIN TEMP2 AND LREL_LATT ON LATT_ID GIVING TEMP3

JOIN TEMP3 AND LREL_LIST ON LREL_ID GIVING TEMP4

JOIN TEMP4 AND SID_LREL ON LREL-ID GIVING TEMP5

PROJECT TEMP5 ON GATT_NAME, SID, DBMS_NAME,
DBMS_TYPE, DB_NAME, LREL_NAME, LATT_NAME,
LREL_INDEX, LREL_REP GIVING DB_RESULT
```

The SELECT operation created a relation TEMP1, containing all the identification codes of the global attributes within the global relation. Next, the first JOIN operation added the unique identification codes (unique keys) of the local attributes which associated with the global attributes to the relation TEMP2. That is, the local attributes were those attributes actually stored in the distributed databases. The following second JOIN operation included the local attribute names that were used in the local databases. The third JOIN operation created a relation TEMP4 which added the information for each local relation in which the local attributes were found. The next JOIN stored the information on the site location of each local relation in the relation TEMP5. Finally, the last PROJECT operation arranged the

attributes in the order that was sent back in the CNDD Data Location Results message.

Whenever a site requested only the data locations of specific global attributes, all the same relational operations, except the SELECT operation, were executed. The SELECT operation was modified to include the name of the global attribute as follows:

```
SELECT GREL_GATT WHERE GREL_NAME = 'RELATION'
AND GATT_NAME = 'ATTRIBUTE' GIVING TEMP1
```

This operation created a relation with the global relation name, global attribute name and identification code of the single attribute requested. After this relational operation, the other operations formed a final relation with all the data locations of only a single global attribute. All of these operations were repeated to find the data locations of each specific global attribute requested.

Normally, the processing which handles a user's query would need the locations of all attributes within a relation so it could optimize how to partition a query. Partitioning a query is deciding how to break up a query into subqueries that are sent to different sites. However, as already explained in Chapter 3, the optimization processing for a PROJECT relational query only needs the locations of those attributes mentioned in the query. It would be unnecessary to know where all the attributes within the global relation were located since the PROJECT operation extracts only the specified attributes from the relation.

In summary, the low level software modules called dBASE II to execute command files which accessed data in the CNDD. These command files were created with a text editor and contained the dBASE II commands necessary to perform the type of relational operations just explained. The next section will describe the software modules implemented to test the ability to request data locations from the CNDD and then retrieve the information from the CNDD.

Partial Implementation of DDBMS

This partial implementation of the DDMBS followed the detailed design described in this thesis and Boeckman's thesis. Because of the magnitude of the DDBMS design, many of the modules mentioned in this section were written as stubs. Later as the DDBMS implementation continues using this top-down programming method, the stubs can be replaced with operational code. In the following structure charts, a circle in the left corner of a module box means the module is a stub, and an asterisk means it was implemented.

Main Executive. The main executive module shown in Figure 12 calls three modules to: initialize the DDBMS, get the next message that has arrived at the site, and start a new process. All of the initialization processing modules were stubs. The "GET_NEXT_MESSAGE" module first gets a local message, one that originated at the same site, if one exists. Local messages were simulated by storing them in a file "LOCAL.TST." If the processing can open the file, it reads

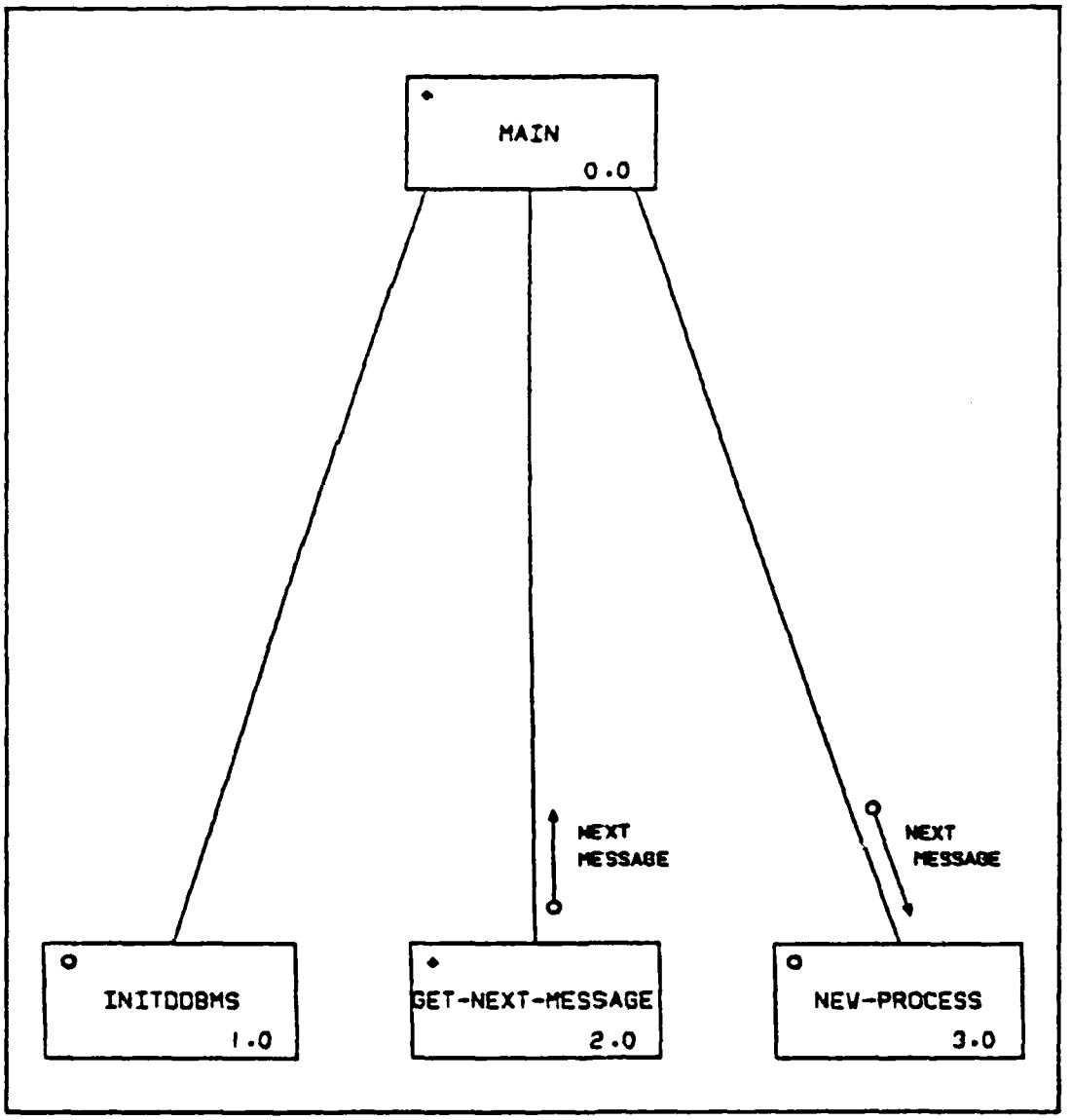


Figure 12. Main Executive

the file and stores the contents into a buffer that is passed to the next dummy module "NEW_PROCESS". If there is no local message, the "GET_NEXT_MESSAGE" module calls "NXT_NETWORK_MESSAGE". This module gets the next network message sent from another site by calling "RECV_FILE", an ISO Layer 6 module in NETOS (Hartrum, 1985:14). However, because of memory limitations, the NETOS software was not loaded on the LSI-11 computer used for the CNDD site. Therefore, network messages were not passed over the LSINET to the CNDD site, but were simulated by reading from the file "REMOTE.TST".

New Process. When "NEW_PROCESS" is implemented with a multi-processing operating system, it will create a process for the message and store it in the process queue. However, in this implementation, the module just calls "DO_PROCESS" as shown in Figure 13. "DO_PROCESS", in turn, calls "INTERPRET" which determines the type of message to process. If it is a reconfiguration type message, the module calls the dummy module "RECONFIGURATION". For all other kinds of messages, it calls the module "REQUESTS". The only module "REQUESTS" calls, which is not a stub module, is "SVC_REQUESTS".

Service Requests. Figure 14 shows that "SVC_REQUESTS" services local requests, remote requests and CNDD requests. The local requests module was implemented because the processing for local requests interrogates the CNDD for data locations. On the contrary, the remote request processing was not implemented because it did not have to access the CNDD.

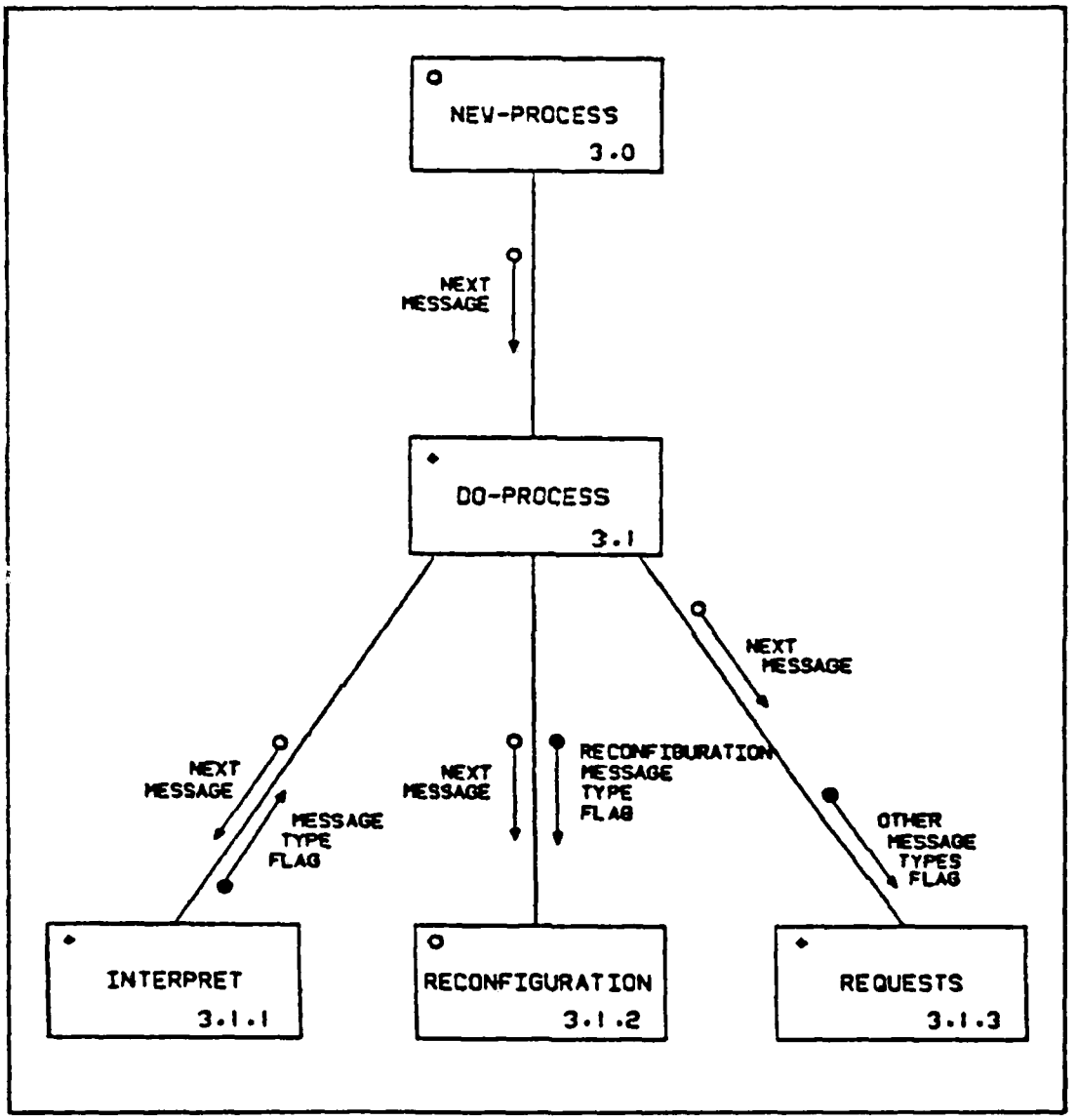


Figure 13. New Process

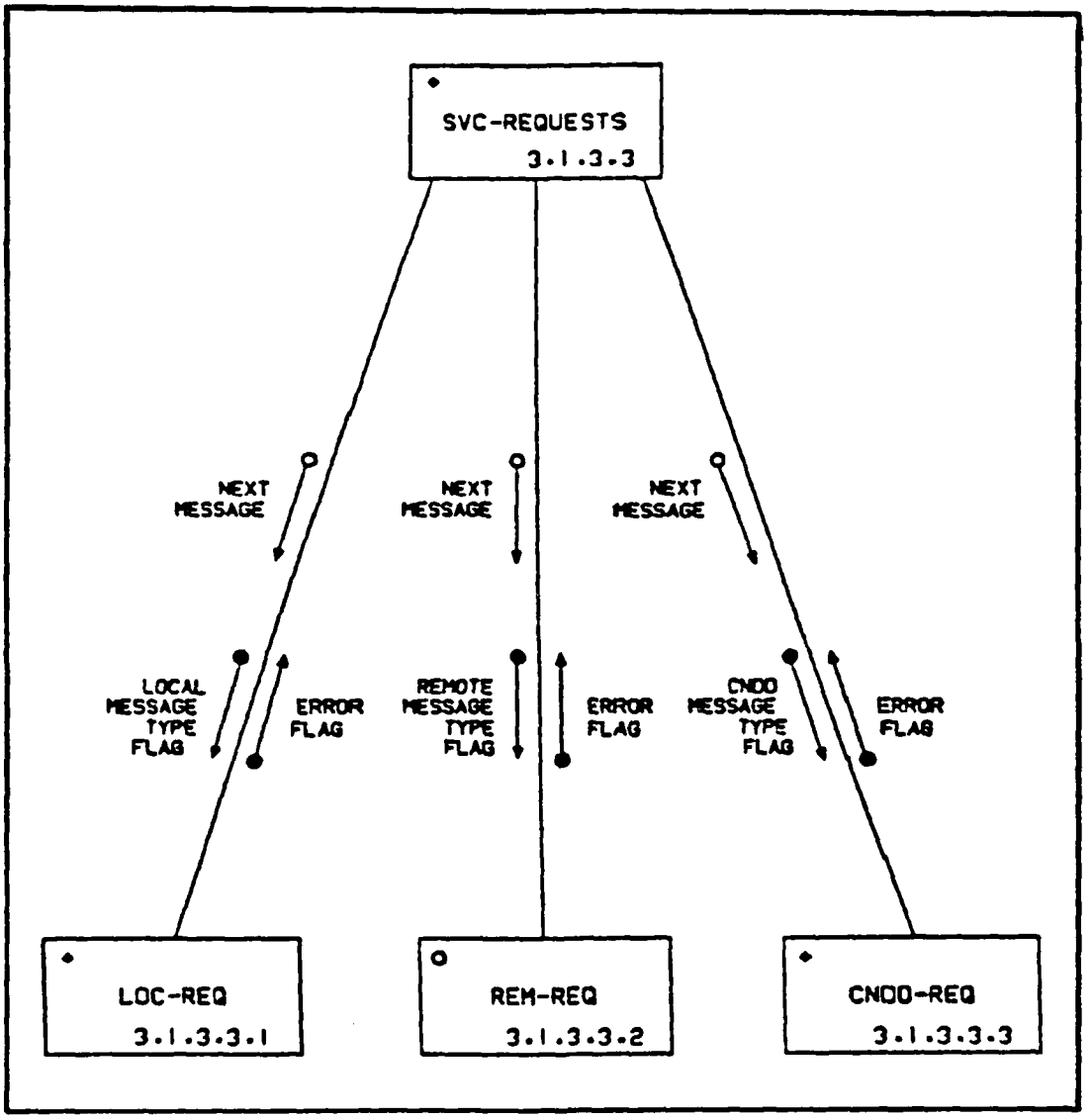


Figure 14. Service Requests

That is, a site would not receive a remote request from another site unless the site contained the data in its host database. Therefore, it would not have to go to the CNDD to find the data location; the data would be in the site's LNDD. The processing for the third type of request, CNDD requests, was partially implemented to achieve the main goal of the thesis.

Service Local Queries. In Figure 15 the local query processing first calls "PARS_QUERY" which parses the query and stores the parts (relations, attributes and conditions) of the query in a data structure. It then calls "DTERMINE_LOCAL_QUERY_TYPE" to decide whether to call either the module "HOST_QUERY" or "NET_QUERY" next. The "HOST_QUERY" module services a query which needs data that is all located on the host computer. In contrast, "NET_QUERY" processes a query where all or some of the data are found at other sites in the network.

Parse Query. As Figure 16 shows, there are three different modules to parse a query written in the Roth relational DB language. "PARS_QUERY" only parses PROJECT, SELECT AND JOIN queries because the translator to convert from the Roth relational language to INGRES only handled these types of queries. Since it was originally planned to connect an INGRES DBMS in the DDBMS, this restricted the kinds of queries that would be processed. However, due to time constraints, the additional software to completely process a query was not

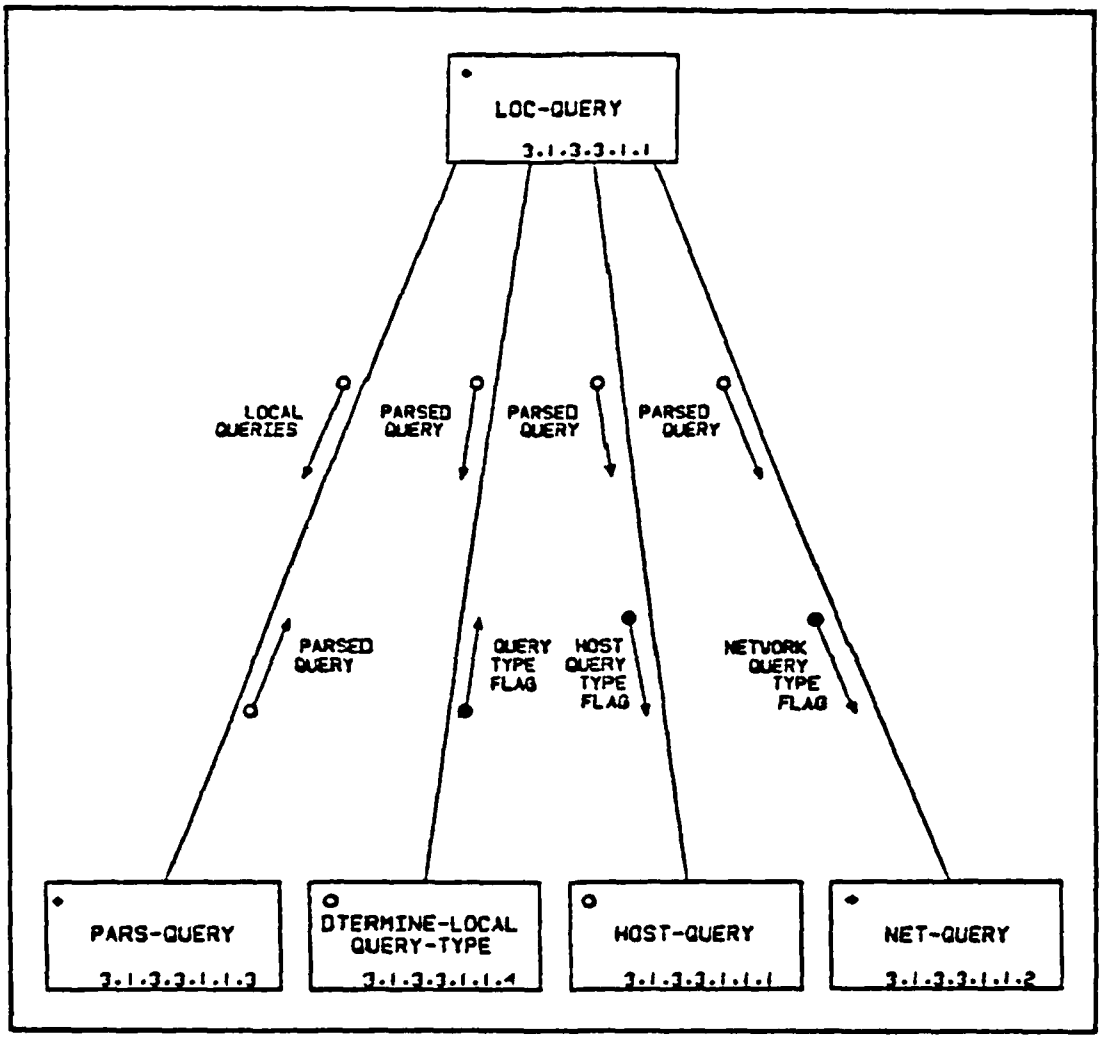


Figure 15. Service Local Queries

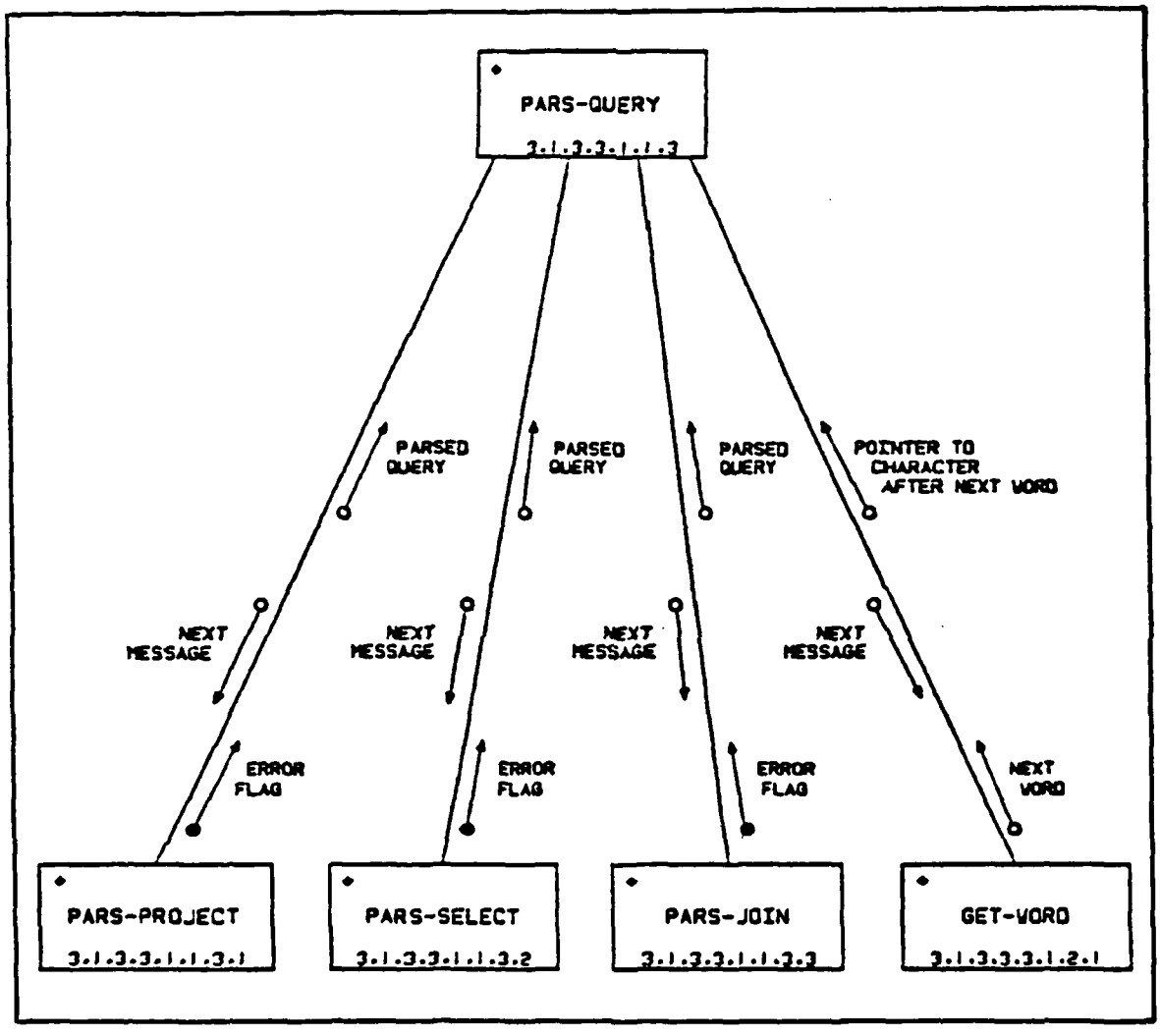


Figure 16. Parse Query

implemented. Therefore, there was no need to connect an INGRES DMBS just to show that the CNDD processing worked.

The parsing modules "PARS_PROJECT", "PARS_SELECT" and "PARS_JOIN" used the procedure "GET_WORD" to read each word of the query. After identifying the relational operation, the relations, the attributes, and the conditions of the query, each of the parsing modules stored the information in the same data structure that Roth used in his implementation (Roth, 1979:54-55). Each relational operator was stored as a node in a tree structure. In this way a query can be partitioned into subqueries, each linked as a node in the tree data structure. The query optimization routines will be able to use this parsed tree structure later when they are implemented. Also, the original query written in the Roth language was retained so that the query processing could use the translators used in Boeckman's thesis (Boeckman, 1984:60-61).

Determine Local Query Type. After the query is parsed, the local query module calls another procedure which determines the query type. To do this, the procedure checks if the locations of the data needed for the query are in the site's LNDD or ECNDD. Because the LNDD and ECNDD were not implemented, this module was coded as a dummy stub. If all the data are located at the host computer, the query type is a host query. Otherwise, the query is classified as a network query. The "HOST_QUERY" module was implemented as a

stub, whereas the "NET_QUERY" module was implemented as shown in Figure 17.

Service Network Queries. To process network queries, the procedure "NET_QUERY" first calls "CHK_CNDD" which interrogates the CNDD for the data locations. If the previous module "DTERMINE_LOCAL_QUERY_TYPE" determined that the locations were not in the LNDD or ECNDD, the CNDD Data Location Request message is built and sent to the CNDD site. The site then waits to receive the CNDD Data Location Results message from the CNDD site.

After checking the CNDD for the data locations, the network query processing calls two dummy modules. Both the modules, "SEND_QUERY_PARTS_TO_REMOTE_LOCATIONS" and "COMPILE_NETWORK_QUERY_RESULTS", were stub modules in this implementation but could be replaced with those written by Boeckman in his partial implementation of the DDBMS. Implementing these modules would complete the network query processing.

Service CNDD Requests. As already explained, Figure 14 showed that the module servicing requests also calls the module "CNDD_REQ" to service CNDD requests, besides the module just explained to service local requests. The only CNDD request implemented was the CNDD Data Location Request. The processing for this request was shown in Figures 6 and 7. Since the processing was implemented as described in Chapter 3, this chapter will not explain the design again.

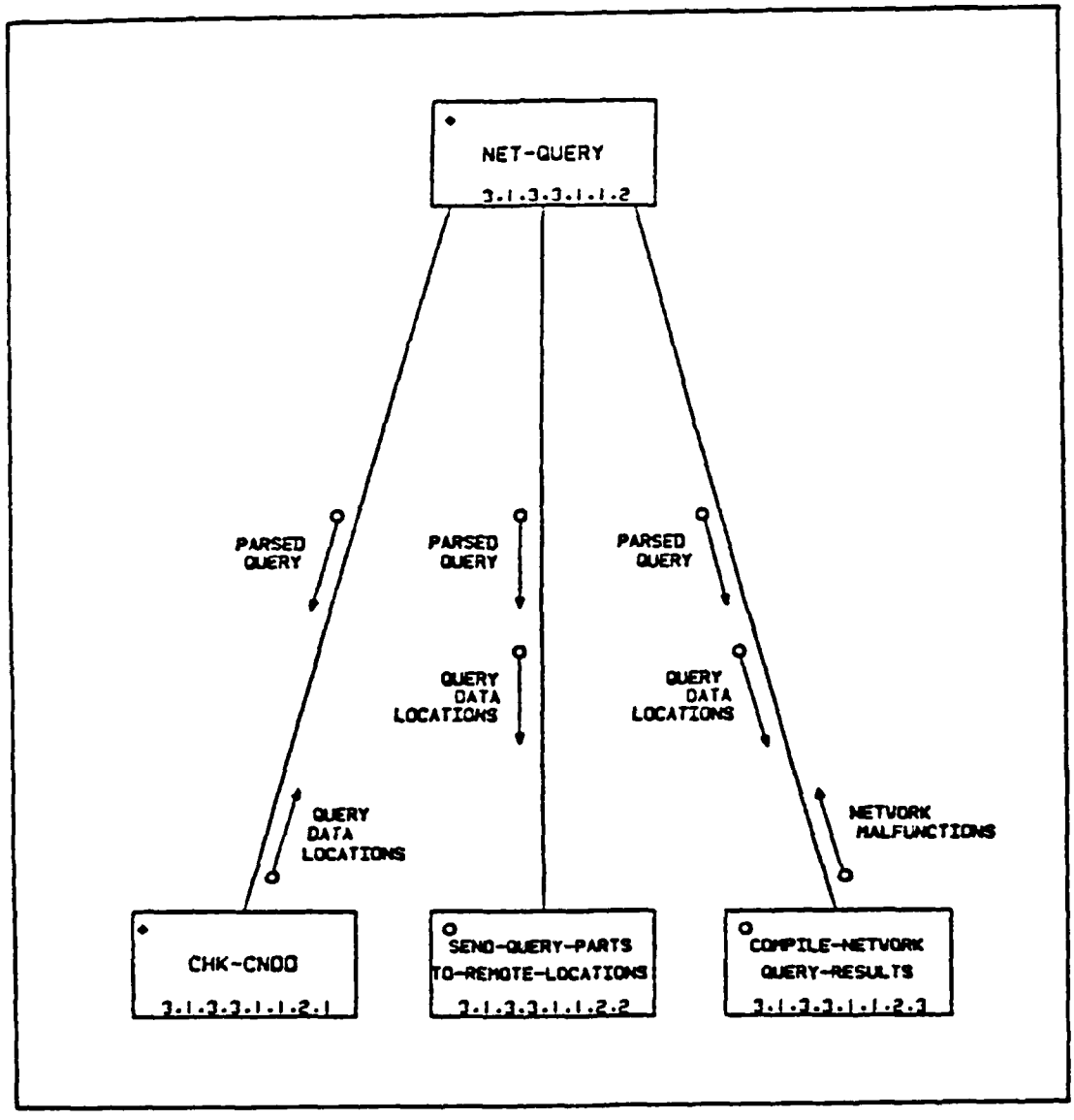


Figure 17. Service Network Queries

Summary

The DDBMS was partially implemented by using two LSI-11 microcomputers and one Z-80-based S-100 bus microcomputer. One of the LSI-11 computers was designated as the DDBMS CNDD site and contained the software to service CNDD Data Location Requests. An S-100 computer, connected to the CNDD site, acted as a host computer to store and access the CNDD with a dBASE II DBMS. The other LSI-11 computer was one of the DDBMS sites and contained the modules to process local network queries. These queries were originated at the site but required data located elsewhere in the DDBMS network. Since all the individual modules of software implemented in this thesis tested successfully, the following chapter explains how the modules were integrated and tested.

V. System Integration Testing

Introduction

In this phase of the thesis project all the software modules implemented were integrated and tested to determine whether they performed together correctly. As the main objective, the testing evaluated the process of requesting and extracting data locations from the CNDD. This involved breaking the testing into two steps:

1) Constructing a CNDD Data Location Request message, and

2) Extracting the information requested from the CNDD and constructing a CNDD Data Location Results message.

To verify these phases, this chapter is divided into three parts. The first section will explain the test data stored in the CNDD. The second section will explain the procedures and results of testing the remote site processing, and the third will cover testing the CNDD site processing.

CNDD Test Data

Two test databases were constructed on different host computers, both of which executed a relational DBMS. A dBASE II DBMS ran on an S-100 microcomputer, and an INGRES DBMS ran on a VAX-11/780 minicomputer. Although the tests did not access these databases through queries, the locations of all the data were stored in the CNDD.

The CNDD maintained a global or conceptual view of the separate databases. This global database, as shown in

Appendix C, contained information about global relations and global attributes. For instance, the user designed DDBMS queries based on these global relation and attribute names. Since this was a distributed DBMS, the global relations and attributes were partitioned or distributed among the databases. Ullman explained that relations can be partitioned either vertically or horizontally (Ullman, 1982:411).

For example, he said if a relation is viewed as a table, vertical partitions represent columns of the table. In other words, a vertically partitioned relation has its attributes--or columns--distributed among several databases.

On the other hand, Ullman explained the horizontally partitioned relation is like a table divided by rows. This means some tuples--or rows--of the relation are located in different databases.

Besides partitioning the attributes of a global relation, the test databases also had duplicate data. The databases copied either columns of attributes or rows of tuples.

Based on these definitions, Figure 18 shows the global relations and how they were partitioned among the two test databases shown in Figures 19a and 19b. A simple naming convention was used to make the mappings more obvious between the global names and the local names used in the actual databases. Except for the first letter, the local relation and local attribute names were the same as the global names with which they corresponded.

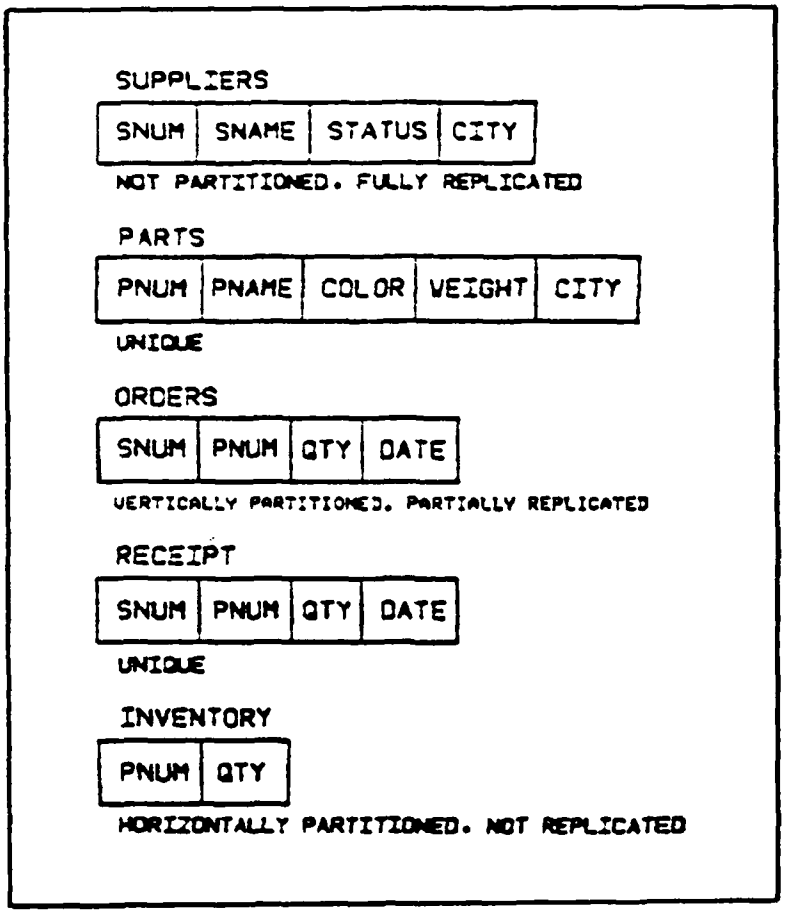


Figure 18. Test Global Relations

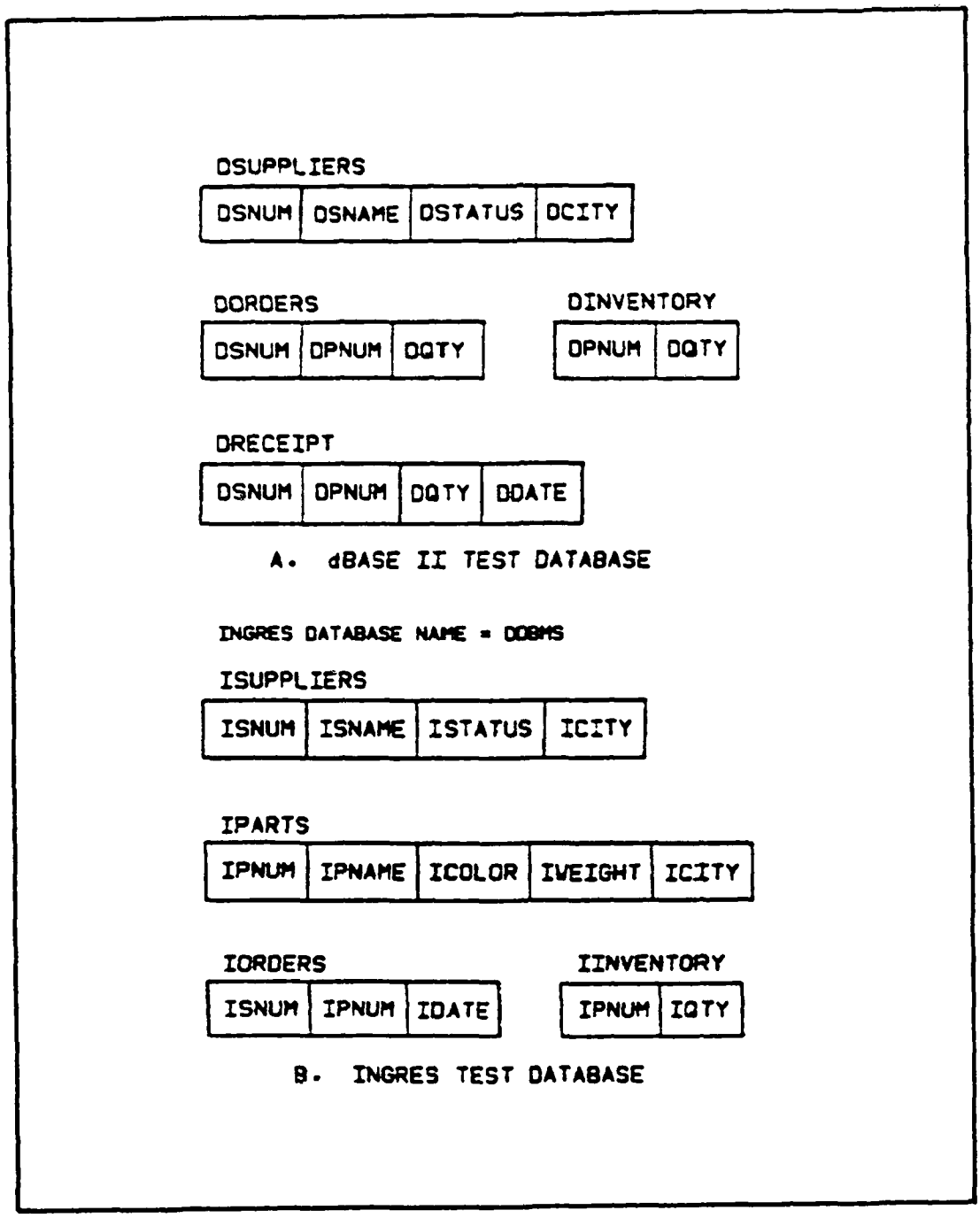


Figure 19. Test DDBMS Database Relations

According to Figure 18 the global relation "SUPPLIERS" was not partitioned, but it was fully replicated at each database. The relations "PARTS" and "RECEIPT" also were not partitioned but were uniquely stored in their entirety at just one of the databases. The relation "ORDERS" was vertically partitioned with some data duplicated at both sites. Finally, the relation "INVENTORY" was horizontally partitioned, and no data was replicated in either database.

Remote Site Processing

During the tests the remote site processing only handled a query up to the point of creating a message that requests data locations to send to the CNDD site. The site did not send the message to the CNDD nor did it send the query to the host databases. The test procedures were as follows.

First, test queries were created with a text editor following the Local Query Request message format in Appendix E and stored in ASCII files (labeled "LOCAL.Q1"..."LOCAL.Q4"). The tests used the queries shown in Figure 20. These queries requested data stored only in one database or in both databases. Also, the PROJECT query included an attribute "bad" which was not part of the global relation. The last query required data from the CNDD that was locked. That is, the CNDD locked access to the relation "inventory" to simulate the data in the CNDD associated with the relation was being updated.

```
Query #1:  SELECT ALL FROM parts
           WHERE (city = 'Chicago') GIVING newrel

Query #2:  JOIN parts, receipt
           WHERE pnum = pnum GIVING newrel

Query #3:  PROJECT suppliers OVER snum, sname, bad
           GIVING newrel

Query #4:  SELECT ALL FROM inventory
           WHERE (pnum = 'PI') GIVING newrel
```

Figure 20. Test Queries

The remote site program "DDBMS" executed four separate runs to test each query. Before each run, the file containing the query was copied into the file "LOCAL.TST". The processing then simulated receiving a local query by reading the file. After parsing the query, the program simulated checking the LNDD and ECNDD for the locations of the relations. This was simulated because neither the LNDD nor the ECNDD was implemented. The processing pretended the data for the relation "receipt" was either in the LNDD or ECNDD.

Finally, the remote site software created a file labeled "A00001.DAT", which contained a CNDD Data Location Request message. For example, the messages for queries #1, 2 and 4 requested the locations of all the global attributes within the global relations "parts", "parts", and "inventory", respectively. The message for query #2 did not include the relation "receipt" because its location was supposedly in the LNDD or ECNDD. In contrast, the request message for query #3 asked for the locations of only the

global attributes "snum", "sname" and "bad" within the relation "suppliers". Test results verified that the request message for each query was built correctly.

CNDD Site Processing

Once the remote site built the CNDD Data Location Request messages, tests checked whether the CNDD site processing extracted the data correctly. Four files (labeled "REMOTE.Q1"..."REMOTE.Q4"), built with a text editor, contained the same request messages that the remote site constructed during its four test runs. Before each run, a file containing the request message was copied into the file "REMOTE.TST". The tests simulated receiving a message from the network by reading the file "REMOTE.TST".

Before starting the program on the LSI-11 computer, the S-100 computer connected to System L was initialized. After cycling up the operating system with the "Super 6 NETOS System Disk", the command "PORTBAUD 9600" was entered to establish a 9600 baud rate. Then the command "Stat con:=crt:" was typed to link the S-100 with the LSI-11 computer through the console port.

After the S-100 was initialized, the program "CNDD" was executed four different times to process each Data Location Request message stored in the file "REMOTE.TST". The software accessed the CNDD and retrieved all the information requested. At the end of the processing, the CNDD site program stored the formatted CNDD Data Results message in a

file named "A00001.DAT". The file "A00001.DAT" was renamed "RESULT.Qx" where x was the query number. After each run, a text editor was used to check that each results message contained the correct format and the data locations as outlined in Figure 19. The fourth results message, though, did not contain any data locations because the access to the data for relation "inventory" was locked. The results of all the CNDD site tests were correct.

Summary

Two test databases were created but were not accessed during the tests. However, the CNDD did contain a directory of where all the data was located. During the first half of the testing, the remote site processing correctly evaluated four test queries and built satisfactory CNDD Data Location Requests. In the last half, the tests verified that the CNDD site correctly supplied the locations of all the data requested. Based on the work during the design, implementation and testing phases of this thesis, the following chapter will discuss ideas for follow-on projects.

VI. Conclusions and Recommendations

Introduction

The previous chapters explain the life cycle process of developing part of the software for the DDBMS central site. During each of the project's phases problems and ideas for future work arose which other individuals may resolve and complete in order to finish an operational DDBMS. This chapter first discusses conclusions about the results of this thesis, and then suggests recommendations for follow-on projects. Finally, the thesis concludes with some final comments.

Conclusions on Results

This project accomplished the main goal of designing the CNDD, implementing it on one of the DDBMS sites, and implementing the software which creates and processes requests for data locations stored in the network CNDD. The integration testing period proved the implemented code worked according to the system requirements and design.

Unfortunately, the DDBMS sites were not connected to a network so that messages could be passed from one site to another. Both the DDBMS software implemented so far and the operating system (NETOS) for the LSINET local area network would not operate on a single LSI-11 microcomputer together. The computer's operating system could not execute all the software in the memory allotted for the program. Conse-

quently, resolving this problem should be the first priority in any future development of the AFIT DDBMS project.

In addition, the thesis described the design of the network messages and the process to update the CNDD, but it did not implement the process. The detailed design also specified the data contents of the LNDD and the ECNDD. However, the project did not implement them nor develop the software which checks for data locations in these local directories because of project time limitations and sizing problems in the LSI-11 computer.

Follow-on Research

The following paragraphs recommend future research based on the results of this thesis and the final goal of implementing an operational DDBMS. The first task should be to find a way to link the DDBMS sites into the LSINET. Other projects include searching the LNDD and ECNDD, updating the ECNDD from CNDD results and implementing the DDBMS on an Intel Hypercube computer. Also, follow-on projects can implement the other CNDD site functions of initializing the DDBMS, reconfiguring the DDBMS, updating the CNDD, and processing pending updates to remote sites. In addition, there are several projects that Boeckman identified in his final chapter (Boeckman, 1984:78-87).

Connecting the DDBMS in a Network. There are several options available to resolve this problem. First, each DDBMS site could connect to another LSI-11 which would contain the

NETOS software and interface with the network. Second, a multi-processing operating system on the LSI-11 could operate both the NETOS and DDBMS software on a single system. Third, all the software could be reorganized into various files to take advantage of overlaying portions of programs over each other in memory. Unless there is a method of accessing more memory with the LSI-11 operating system, the sizing problem will occur over and over.

Implementing the LNDD and ECNDD. Now that the CNDD is implemented, the other directories should be implemented. Using the definitions of the LNDD and ECNDD contents described in this study and the DDBMS overall design, follow-on research can refine the detailed design of these directories. Mahoney's work on the global translator contains some of the mapping information needed that should be added to the LNDD contents described in this thesis (Mahoney, 1985). After implementing the directories, the research should implement the modules to search these additional directories for data locations and to update the ECNDD when a site receives CNDD results.

Implementing the DDBMS on Intel Hypercube. Since the AFIT DEL plans to receive an Intel Hypercube computer, another future project may implement the DDBMS on this multiprocessor computer. Several parts of the DDBMS software may be hosted on some of the 32 processors in this system.

Initializing the DDBMS. The current implementation does not dynamically check which sites are connected in the DDBMS. Instead, a table stored in a ASCII file contains default status values of all possible sites in the LSINET. A future project could replace this table by implementing the initial contact and startup messages as designed.

Reconfiguring the DDBMS. A follow-on thesis could write more implementation-specific structure charts and implement the ability to reconfigure the system. This includes the processing to add or delete a remote site and to move the CNDD to a new central site. If the CNDD at the new site is not implemented as done in this thesis, the lower level modules coded in this thesis, which extract data from the dBASE II database, must be redone.

Updating the CNDD. Using the design explained in this thesis, a follow-on effort could implement the messages and the processing required to update the CNDD. This project would probably design more implementation-specific structure charts before coding the modules.

Processing Pending Updates. Before the CNDD site can process pending updates to inactive remote sites, the ability to update the databases must be added to the current Roth-dBASE II and Roth-INGRES translators or included in new translators. As Boeckman suggested (Boeckman, 1984:80), the translators could use the EDIT commands of Roth (Roth, 1979:119-121) and convert them to appropriate commands in

INGRES and dBASE II. Besides changing the translators, the researcher must also implement some update concurrency algorithm. After implementing the update capability, the follow-on project could refine the design and implement the pending update processing at the CNDD site.

Other DDBMS Projects. In order to complete the DDBMS implementation, researchers must complete several other projects that Boeckman identified (Boeckman, 1984:78-87). For example, designing and implementing a query optimization algorithm is necessary to be able to efficiently process input queries. This includes partitioning a query into subqueries, routing the queries to the optimum sites and computing the query results. Another project could design and implement queue processing algorithms for the network messages if a multi-processing operating system is used in the DDBMS. This would speed up the DDBMS processing. For example, a site could receive several messages, which would be stored in a queue, at the same time as it was processing the highest priority message. Also, the site could store all output messages in another queue and continue its processing without having to wait for the network operating system to send each message to its destination.

Final Comments

Future work on the AFIT DDBMS should concentrate on connecting the DDBMS in a network, first and foremost, and then initializing the DDBMS, updating the CNDD, processing

APPENDIX A

CNDD DATA DEFINITIONS

Field Name	Field Definition	Possible Values	Description
grel_name	15 chars		Unique global relation name
grel_access	1 digit		Lock to prevent access to any of the global relation's data during update
		Ø	Locked (no access)
		1	Unlocked
gatt_id	15 chars		Unique global attribute id
gatt_name	15 chars		Global attribute name (*does* not have to be unique)
sid	1Ø chars		Logical site id of system connected to the DDBMS network
host	3 chars		Type of host computer (contains a *DBMS*) which is connected to another processor connected to the DDBMS network
		CDC	CDC Cyber
		1ØØ	S-1ØØ
		UNX	VAX 11/78Ø with UNIX o/s
		VMS	*VAX 11/78Ø with VMX o/s*
dbms_name	3 chars		Name of Data Base Management System (DBMS) on host computer
		DBT	DBTG
		ING	Ingres
		DB2	*dBase II*
		TOT	Total
		IMS	IMS
dbms_type	1 char		Type of DBMS on host
		H	Hierarchical
		N	Network
		R	Relational

db_name	15 chars		Name of Database (DB) on host computer
lrel_id	15 chars		Unique local relation id
lrel_name	15 chars		Local relation name unique only in host computer DB
lrel_index	1 digit		Local relation index code
		Ø	Not indexed on an attribute
		1	Indexed on an attribute
lrel_access	1 digit		Lock to prevent access to local relation's data during update
		Ø	Locked (no access)
		1	Unlocked
lrel_rep	2 digits		Local relation replication code
		1	No partitioning with no redundancy
		2	No partitioning with complete redundancy
		3	Vertically partitioned[1] with partial redundancy[2]
		4	Vertically partitioned with no redundancy
		5	Horizontally partitioned[3] with no redundancy
		6	Horizontally partitioned with partial redundancy
		7	Vertically and horizontally partitioned[4] with no redundancy
		8	Vertically and horizontally partitioned with partial redundancy
		9	Horizontally and vertically partitioned[5] with no redundancy

		10	Horizontally and vertically partitioned with partial redundancy
latt_id	15 chars		Unique local attribute id
latt_name	15 chars		Local attribute name (does not have to be unique)
latt_access	1 digit		Lock to prevent access to local attribute's data during update
		0	Locked (no access)
		1	Unlocked

[1] According to Ullman (Ullman, 1982:411), vertical partitioning is when the partitions are columns of the relational table. That is, the attributes of the global relation are in different local relations.

[2] Redundancy means some of the data in different local relations is duplicated.

[3] Horizontal partitioning separates the table (relation) by rows (tuples). In other words, each tuple of a local relation contains all the attributes of the global relation, but no local relation contains all the tuples of data.

[4] There are at least two vertical partitions, one or more of which is further divided into horizontal partitions.

[5] There are at least two horizontal partitions, one or more of which is further divided into vertical partitions.

APPENDIX B

CNDD USER'S GUIDE

Introduction

This User's Guide explains how to maintain the Centralized Network Data Directory (CNDD) of the DDBMS. The CNDD was implemented as a database itself using the dBASE II DBMS which executed on a Z-80-based S-100 bus microcomputer. Appendix A defines the data items in the CNDD, and Appendix C shows the CNDD test database implemented to evaluate the DDBMS performance. This appendix describes the procedures and the dBASE II commands necessary for changing the CNDD.

Initialization Data

In this implementation of the DDBMS the initialization modules were not implemented. These modules ask the operator which site is the CNDD site and which sites are part of the DDBMS network. Because of not implementing the initialization processing, this version of the DDBMS used an ASCII file containing this information. The file, called "DTABLE.DAT", contained the three-letter designator of the CNDD site followed by a line feed (LF) on the first line. The following lines list the designator of each site in the LSINET and the site's status, with a LF after the designator and the status code. The status code is "1" if the site is connected in the DDBMS network and "0" if it is not connected.

If the CNDD site changes from System L (three-letter designator "LSL") as implemented now, the first line must change to reflect the new designator. For example, the database administrator (DBA) must use a text editor to change the first line to "LSS" if System S is selected as the new CNDD site.

Changing CNDD Data

When the CNDD data changes, the DBA must use dBASE II to change the relations shown in Appendix C. To make the changes, the S-100 disk drive 0 must contain the diskette labeled "DDBMS System Disk", and drive 1 must contain the diskette labeled "DDBMS Data Disk". Begin the dBASE II DBMS by typing "dbase<return>" and then the date followed by <return>. The following procedures outline the process to add to the CNDD the data pertaining to a new local relation stored at one of the host databases. This example was used because it showed how to change all the relations of the CNDD.

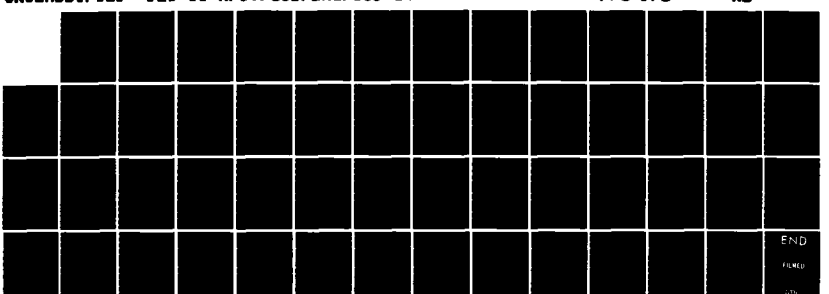
Adding Data. For instance, the DBA wants to add the local relation called "ireceipt" to the host database connected to System K. The local relation has the local attributes "isnum", "ipnum" and "iqty".

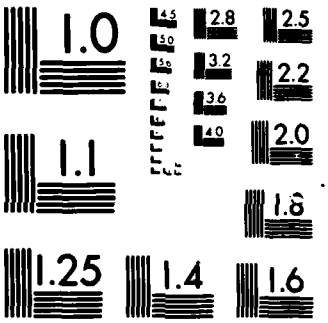
First, the DBA decides the local relation is part of the global relation called "receipt". Now, he changes the CNDD relation called "grellrel" by typing:

```
USE grellrel<return>
APPEND<return>
```

AD-A163 843 DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY FOR A DISTRIBUT..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J A WEDERTZ 2/2

UNCLASSIFIED DEC 85 AFIT/GCS/ENG/85D-24 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

dBASE II will now expect the DBA to enter the attributes of the relation "grellrel", namely the global relation name, the access code, and the local relation id. The access code will be "1" and the unique local relation id will be "ireceipt". Enter the data by typing:

```
receipt<return>
1
ireceipt<return>
```

When "1" is typed, the computer will make a beep, and the cursor will automatically jump to the next field without typing the <return>. Now type <control Q> to stop appending to the relation "grellrel". If the DBA wants to examine the data, he types "LIST<return>".

Next, the DBA decides the local attributes "isnum", "ipnum" and "iqty" match with the global attributes "snum", "pnum" and "qty", respectively. Also, the DBA determines a unique global attribute id for each global attribute. These ids will be "recsnum", "recpnum" and "recqty". Since this information is already in the relation "grelgatt", nothing must be appended. However, if the information were not in the CNDD relation "grelgatt", the DBA would type:

```
USE grelgatt<return>
APPEND<return>
receipt<return>
snum<return>
recsnum<return>
receipt<return>
pnum<return>
recpnum<return>
receipt<return>
qty<return>
recqty<return>
<CONTROL Q>
```

The DBA next adds the information for the CNDD relation "sidlrel". This data includes the site id, the host name, the DBMS name and type, the DB name and the local relation id. In this example, the site id is "LSK", the host name is "UNX" (see Appendix A), the DBMS name is "ING", the DBMS type is "R", the DB name is "ddbms" and the local relation id is "ireceipt". This information is already in the CNDD database, but if were not in the database type:

```
USE sidlrel<return>
APPEND<return>
LSK<return>
UNX
ING<return>
R
ddbms<return>
ireceipt<return>
<CONTROL Q>
```

The next CNDD relation "lrellist" contains information about the local relation. This data includes the local relation id, name, index code, access code and replication code. After reviewing Appendix A, the DBA decides the index code is "0", the access code is "1" and the replication code is "5". In this example both the local relation id and name are the same. To enter the data type:

```
USE lrellist<return>
APPEND<return>
ireceipt<return>
ireceipt<return>
0
1
5<return>
(CONTROL Q>
```

The DBA now appends data about the local attributes. This includes the local relation id and the local attribute

id, name and access code. In order to make the local attribute ids unique, the DBA assigns the ids "irecsnum", "irecpnum" and "irecqty" to the local attributes "isnum", "ipnum" and "iqty", respectively. He also assigns an access code of "1" to each attribute. To enter the data type:

```
USE lrellatt<return>
APPEND<return>
ireceipt<return>
irecsnum<return>
isnum<return>
1
ireceipt<return>
irecpnum<return>
ipnum<return>
1
ireceipt<return>
irecqty<return>
iqty<return>
1
<CONTROLQ>
```

Finally, the DBA matches the global and local attributes to each other in the CNDD relation "gattlatt". This CNDD relation contains the global attribute id and the local attribute id. The DBA enters the following:

```
USE gattlatt<return>
APPEND<return>
recsnum<return>
irecsnum<return>
recpnum<return>
irecpnum<return>
recqty<return>
irecqty<return>
<CONTROL Q>
```

Examining Data. If the DBA wants to examine the data in any CNDD relation, he opens the CNDD relation and lists the data. For example, if he wants to check the data in the relation "gattlatt", he types:

```
USE gattlatt<return>
LIST<return>
```

Correcting Data. If the DBA finds an error in one of the relation's records, he must note the record number of the bad record. For example, if record #3 in relation "grellrel" has an error, the DBA types:

```
USE grellrel<return>
EDIT<return>
```

The program will display:

```
ENTER RECORD #:
```

Enter:

```
3<return>
```

dBASE II will clear the screen and then display all the data in record #3. The DBA can then correct data in any field by typing over the incorrect field. If a field within a record is correct, type <return> to move to the next field.

Deleting Data. If the DBA decides to delete record #4 from the relation "grelgatt", for example, he types:

```
USE grelgatt<return>
DELETE RECORD 4<return>
```

This record is only marked with a flag for deletion and is not actually removed from the relation. If the DBA wants to unmark the deletion flag he types:

```
RECALL RECORD 4<return>
```

If the DBA wants to remove the record, he uses the PACK command. After executing this command, the DBA cannot recall a record.

APPENDIX C

CNDD Test Database

GREL_LREL

GREL_NAME	GREL_ACCESS	LREL_ID
suppliers	1	isuppliers
suppliers	1	dsuppliers
parts	1	iparts
orders	1	iorders
orders	1	dorders
receipt	1	dreceipt
inventory	0	iinventory
inventory	0	dinventory

GREL_GATT

GREL NAME	GATT NAME	GATT ID
suppliers	snum	supsnum
suppliers	sname	supsname
suppliers	status	supstatus
suppliers	city	supcity
parts	pnum	parpnum
parts	pname	parpname
parts	color	parcolor
parts	weight	parweight
parts	city	parcity
orders	snum	ordsnum
orders	pnum	ordpnum
orders	qty	ordqty
orders	date	orddate
receipt	snum	recsnum
receipt	pnum	recpnum
receipt	qty	recqty
receipt	date	recdate
inventory	pnum	invpnum
inventory	qty	invqty

SID_LREL

SITE ID	HOST NAME	DBMS NAME	DBMS TYPE	DB NAME	LREL ID
LSK	UNX	ING	R	ddbms	isuppliers
LSK	UNX	ING	R	ddbms	iparts
LSK	UNX	ING	R	ddbms	iorders
LSK	UNX	ING	R	ddbms	iinventory
LSS	100	DB2	R		dsuppliers
LSS	100	DB2	R		dorders
LSS	100	DB2	R		dreceipt
LSS	100	DB2	R		dinventory

LREL_LIST

LREL ID	LREL NAME	LREL INDEX	LREL ACCESS	LREL REP
isuppliers	isuppliers	0	1	2
iparts	iparts	0	1	1
iorders	iorders	0	1	3
iinventory	iinventory	0	1	5
dsuppliers	dsupplie	0	1	2
dorders	dorders	0	1	3
dreceipt	dreceipt	0	1	1
dinventory	dinvento	0	0	5

LREL_LATT

LREL ID	LATT ID	LATT NAME	LATT ACCESS
isuppliers	isupsnum	isnum	1
isuppliers	isupsname	isname	1
isuppliers	isupstatus	istatus	1
isuppliers	isupcity	icity	1
iparts	iparpnum	ipnum	1
iparts	iparpname	ipname	1
iparts	iparcolor	icolor	1
iparts	iparweight	iweight	1
iparts	iparcity	icity	1
iorders	iordsnum	isnum	1
iorders	iordpnum	ipnum	1
iorders	iorddate	idate	1
iinventory	iinvpnum	ipnum	1
iinventory	iinvqty	iqty	1
dsuppliers	dsupsnum	dsnum	1
dsuppliers	dsupsname	dsname	1
dsuppliers	dsupstatus	dstatus	1
dsuppliers	dsupcity	dcity	1
dorders	dordsnum	dsnum	1
dorders	dordpnum	dpnum	1
dorders	dordqty	dqty	1
dreceipt	drecsnum	dsnum	1
dreceipt	drecpnum	dpnum	1
dreceipt	drecqty	dqty	1
dreceipt	drecdate	ddate	1
dinventory	dinvpnum	dpnum	0
dinventory	dinvqty	dqty	0

GATT_LATT

GATT ID	LATT ID
invpnum	iinvpnum
invpnum	dinvpnum
invqty	iinvqty
invqty	dinvqty
orddate	iorddate
ordpnum	iordpnum
ordpnum	dordpnum
ordqty	dordqty
ordsnum	iordsnum
ordsnum	dordsnum
parcity	iparcity
parcolor	iparcolor
parpname	iparpname
parpnum	iparpnum
parweight	iparweight
recdate	drecdate
recpnum	drecpnum
recqty	drecqty
recsnum	drecsnum
supcity	isupcity
supcity	dsupcity
supsname	isupsname
supsname	dsupsname
supsnum	isupsnum
supsnum	dsupsnum
supstatus	isupstatus
supstatus	dsupstatus

APPENDIX D

LNDD DATA DEFINITIONS

Field Name	Field Definition	Possible Values	Description
grel_name	15 chars		Unique global relation name
grel_access	1 digit		Lock to prevent access to any of the global relation's data during update
		Ø	Locked (no access)
		1	Unlocked
gatt_id	15 chars		Unique global attribute id
gatt_name	15 chars		Global attribute name (does not have to be unique)
host	3 chars		Type of host computer (contains a DBMS) which is connected to another processor connected to the DDBMS network
		CDC	CDC Cyber
		100	S-100
		UNX	VAX 11/780 with UNIX o/s
		VMS	VAX 11/780 with VMX o/s
dbms_name	3 chars		Name of Data Base Management System (DBMS) on host computer
		DBT	DBTG
		ING	Ingres
		DB2	dBase II
		TOT	Total
		IMS	IMS
dbms_type	1 char		Type of DBMS on host computer
		H	Hierarchical
		N	Network
		R	Relational
db_name	15 chars		Name of Database (DB) on host computer

lrel_id	15 chars		Unique local relation id
lrel_name	15 chars		Local relation name unique only in host computer DB
lrel_index	1 digit		Local relation index code
		0	Not indexed on an attribute
		1	Indexed on an attribute
lrel_access	1 digit		Lock to prevent access to local relation's data during update
		0	Locked (no access)
		1	Unlocked
lrel_rep	2 digits		Local relation replication code
		1	No partitioning with no redundancy
		2	No partitioning with complete redundancy
		3	Vertically partitioned[1] with partial redundancy[2]
		4	Vertically partitioned with no redundancy
		5	Horizontally partitioned[3] with no redundancy
		6	Horizontally partitioned with partial redundancy
		7	Vertically and horizontally partitioned[4] with no redundancy
		8	Vertically and horizontally partitioned with partial redundancy
		9	Horizontally and vertically partitioned[5] with no redundancy

		10	Horizontally and vertically partitioned with partial redundancy
latt_id	15 chars		Unique local attribute id
latt_name	15 chars		Local attribute name (does not have to be unique)
latt_access	1 digit		Lock to prevent access to local attribute's data during update
		0	Locked (no access)
		1	Unlocked
seg_name	15 chars		Segment name
seg_size	4 digits		Segment size
seg_seq	4 digits		Segment sequence number
field_name	15 chars		Field name
field_size	4 digits		Field size
field_type	1 char		Field type
		N	Numeric
		C	Character
par_name	15 chars		Parent name
chd_name	15 chars		Child name
set_name	15 chars		Set name
set_type	1 char		Set type
		N	Numeric
		C	Char
rec_name	15 chars		Record name
item_name	15 chars		Item name
item_type	1 char		Item type
		N	Numeric
		C	Char
item_len	4 digits		Item length

sort	1 digit		Sort code
		Ø	Not sorted
		1	Sorted
sort_key	15 chars		Sort key name
sort_order	1 char		Sort order
		A	Ascending
		D	Descending

[1] According to Ullman (Ullman, 1982:411), vertical partitioning is when the partitions are columns of the relational table. That is, the attributes of the global relation are in different local relations.

[2] Redundancy means some of the data in different local relations is duplicated.

[3] Horizontal partitioning separates the table (relation) by rows (tuples). In other words, each tuple of a local relation contains all the attributes of the global relation, but no local relation contains all the tuples of data.

[4] There are at least two vertical partitions, one or more of which is further divided into horizontal partitions.

[5] There are at least two horizontal partitions, one or more of which is further divided into vertical partitions.

APPENDIX E

MESSAGE FORMATS

Description

This appendix shows the format for messages transferred over the network in this implementation of the DDBMS. This is a subset of those messages which Boeckman designed (Boeckman, 1984:Appendix C) that deal with the directory system. Changes from the original Boeckman design were necessary because of the methods of implementation.

CNDD Data Location Request

Field No.	Field Definition	Value	Description
1	1 char	STX	Start of message
2	3 chars	CDL	Message type
3	1 char	LF	Field delimiter
4	10 chars		System ID at destination computer
5	1 char	LF	Field delimiter
6	10 chars		System ID at source computer
7	1 char	LF	Field delimiter
8	4 chars		Unique process ID
9	1 char	LF	Field delimiter
10	10 chars		Time stamp (HH:MM:SS.T)
11	1 char	LF	Field delimiter
12	10 chars		Password
13	1 char	LF	Field delimiter
14	1 digit		Location Type Request Code
		1	Request locations of all attributes within the following global relation; no attribute names listed immediately after the following relation name
		2	Request locations of some attributes within the following global relation; the global attribute names listed after the global relation name - see description in field 18
15	1 char	LF	Field delimiter
16	15 chars		Global relation name*
17	1 char	LF	Field delimiter

18	Varies		a. If field 14 contains "1", repeat fields 14-18 until all relations and attributes are listed, or b. If field 14 contains "2", list only the names* of the global attributes within the previous global relation for which locations requested. Place <LF> after each name. When the attribute list is complete, repeat fields 14-18 until all relations and attributes are listed.
N	1 char	ETX	End of message (N = last field)

Example of Fields 14-N

1<LF>
student<LF>
2<LF>
faculty<LF>
name<LF>
address<LF>
1<LF>
staff<LF>
<ETX>

The CNDD will send the locations of all the attributes within the relations student and staff and only the locations of the attributes name and address within the relation faculty.

* If the length of a name is less than its maximum size, a LF is placed immediately after the names without any padded blanks before the LF.

CNDD Data Location Results

Field No.	Field Definition	Value	Description
1	1 char	STX	Start of message
2	3 chars	CDR	Message type
3	1 char	LF	Field delimiter
4	10 chars		System ID at destination computer
5	1 char	LF	Field delimiter
6	10 chars		System ID at source computer
7	1 char	LF	Field delimiter
8	4 chars		Unique process ID
9	1 char	LF	Field delimiter
10	10 chars		Time stamp (HH:MM:SS.T)
11	1 char	LF	Field delimiter
12	2 chars	R=	Relation name in next field
13	1 char	LF	Field delimiter
14	15 chars		Global relation name*
15	1 char	LF	Field delimiter
16	2 chars	A=	Attribute name in next field
17	1 char	LF	Field delimiter
18	15 chars		Global attribute name*
19	1 char	LF	Field delimiter
20	2 chars	L=	Data location in next field
21	1 char	LF	Field delimiter
22	10 chars		System ID where data located, or
		0	Data not found anywhere in DDBMS; skip to field 38b; do not fill in the following fields, or

		1	Access locked to data being updated
23	1 char	LF	Field delimiter
24	3 chars		DBMS name
		DBT	DBTG
		ING	Ingres
		DB2	dBASE II
		TOT	Total
		IMS	IMS
25	1 char	LF	Field delimiter
26	1 digit		DBMS type
		H	Hierarchical
		N	Network
		R	Relational
27	1 char	LF	Field delimiter
28	15 chars		Database name*
29	1 char	LF	Field delimiter
30	15 chars		Local relation name*
31	1 char	LF	Field delimiter
32	15 chars		Local attribute name*
33	1 char	LF	Field delimiter
34	1 digit		Index Code
		Ø	Local relation not indexed on a field
		1	Local relation indexed on a field
35	1 char	LF	Field delimiter
36	2 digits		Replication code
		1	No Partitioning with No Redundancy (unique local relation contains all attributes of global relation, and data are in only one place)
		2	No Partitioning with Complete Redundancy (local relation contains all attributes of global relation,

			but data are fully replicated in at least one other place)
		3	Vertically Partitioned with No Redundancy (different subsets of global attributes within global relation in one or more local relations, but no data and non-key attributes are redundant)
		4	Vertically Partitioned with Partial Redundancy (same as 3, except some data and non-key attributes are redundant)
		5	Horizontally Partitioned with No Redundancy (several local relations contain all attributes of global relation, but no data in any relation are redundant)
		6	Horizontally Partitioned with Partial Redundancy (same as 5, except some data are redundant)
		7	Vertically & Horizontally Partitioned with No Redundancy (global relation contains two or more vertical partitions, one or more of which further divided into horizontal partitions; no data are redundant)
		8	Vertically & Horizontally Partitioned with Partial Redundancy (same as 7, except horizontal, vertical or both partitions have redundant data)
		9	Horizontally & Vertically Partitioned with No Redundancy (global relation contains two or more horizontal partitions, one or more of which further divided into vertical partitions; no data are redundant)
		10	Horizontally & Vertically Partitioned with Partial Redundancy (same as 9, except horizontal, vertical or both partitions have redundant data)
37	1 char	LF	Field delimiter

38	Varies		a. Repeat information in fields 20-37 for each local relation-local attribute pair that associates with the global relation-global attribute pair. b. When there are no more locations to list for this global attribute, list another global attribute within the global relation as in fields 16-19. Then repeat step a and this step until there are no more global attributes to list within this global relation. c. List another global relation as in fields 12-15. Then repeat step b and this step until there are no more global relations to list.
N	1 char	ETX	End of message (N = last field)

* If the length of a name is less than its maximum size, A LF is placed immediately after the names without any padded blanks before the LF.

CNDD Update Message to ECNDD
and
LNDD Updates from CNDD

Field No.	Field Definition	Value	Description
1	1 char	STX	Start of message
2	3 chars		Message type
		CUM	CNDD Update Message to ECNDD
		LUC	LNDD Updates from CNDD
3	1 char	LF	Field delimiter
4	10 chars		System ID at destination computer
5	1 char	LF	Field delimiter
6	10 chars		System ID at source computer
7	1 char	LF	Field delimiter
8	4 chars		Unique process ID
9	1 char	LF	Field delimiter
10	10 chars		Time stamp (HH:MM:SS.T)
11	1 char	LF	Field delimiter
12	1 char		Update type
		A	Add
		D	Delete
		M	Modify
13	1 char	LF	Field delimiter

If update type is delete or modify, fields 14-33 must contain the old values which are used as a combined key to locate the data. Only fields 34-54 contain the modified values.

14	15 chars		Global relation name*
15	1 char	LF	Field delimiter
16	15 chars		Global attribute name*
17	1 char	LF	Field delimiter

18	10 chars		System ID where data stored
19	1 char	LF	Field delimiter
20	3 chars		DBMS name
		DBT	DBTG
		ING	Ingres
		DB2	dBASE II
		TOT	Total
		IMS	IMS
21	1 char	LF	Field delimiter
22	1 char		DBMS type
		H	Hierarchical
		N	Network
		R	Relational
23	1 char	LF	Field delimiter
24	15 chars		Database name*
25	1 char	LF	Field delimiter
26	15 chars		Local relation name*
27	1 char	LF	Field delimiter
28	15 chars		Local attribute name*
29	1 char	LF	Field delimiter
30	1 digit		Index Code
		0	Local relation not indexed on a field
		1	Local relation indexed on a field
31	1 char	LF	Field delimiter
32	2 digits		Replication code (see description in CNDD Data Location Results message)
33	1 char	LF	Field delimiter

For Add or Delete Update Type:

34	1 char	ETX	End of message; do not fill in the following fields

For Modify Update Type:

List only the modified values in the following fields. Put a single blank in any field not modified.

34	15 chars		Global relation name*
35	1 char	LF	Field delimiter
36	15 chars		Global attribute name*
37	1 char	LF	Field delimiter
38	10 chars		System ID where data stored
39	1 char	LF	Field delimiter
40	3 chars		DBMS name
41	1 char	LF	Field delimiter
42	1 char		DBMS type
43	1 char	LF	Field delimiter
44	15 chars		Database name*
45	1 char	LF	Field delimiter
46	15 chars		Local relation name*
47	1 char	LF	Field delimiter
48	15 chars		Local attribute name*
49	1 char	LF	Field delimiter
50	1 digit		Index Code
51	1 char	LF	Field delimiter
52	2 digits		Replication code
53	1 char	LF	Field delimiter
54	1 char	ETX	End of message

* If the length of a name is less than its maximum size, a LF is placed immediately after the names without any padded blanks before the LF.

CNDD Updates and External LNDD Updates

Field No.	Field Definition	Value	Description
1	1 char	STX	Start of message
2	3 chars		Message type
		CUP	CNDD Updates
		ELU	External LNDD Updates
3	1 char	LF	Field delimiter
4	10 chars		System ID at destination computer
5	1 char	LF	Field delimiter
6	10 chars		System ID at source computer
7	1 char	LF	Field delimiter
8	4 chars		Unique process ID
9	1 char	LF	Field delimiter
10	10 chars		Time stamp (HH:MM:SS.T)
11	1 char	LF	Field delimiter
12	1 char		Update type
		A	Add
		D	Delete
		M	Modify
13	1 char	LF	Field delimiter

If update type is delete or modify, fields 14-29 must contain the old values which are used as a combined key to locate the data. Only fields 30-46 contain the modified values.

14	10 chars		System ID where data stored
15	1 char	LF	Field delimiter
16	3 chars		DBMS name
		DBT	DBTG
		ING	Ingres

		DB2	dBASE II
		TOT	Total
		IMS	IMS
17	1 char	LF	Field delimiter
18	1 char		DBMS type
		H	Hierarchical
		N	Network
		R	Relational
19	1 char	LF	Field delimiter
20	15 chars		Database name*
21	1 char	LF	Field delimiter
22	15 chars		Local relation name*
23	1 char	LF	Field delimiter
24	15 chars		Local attribute name*
25	1 char	LF	Field delimiter
26	1 digit		Index Code
		Ø	Local relation not indexed on a field
		1	Local relation indexed on a field
27	1 char	LF	Field delimiter
28	2 digits		Replication code (see description in CNDD Data Location Results message)
29	1 char	LF	Field delimiter

For Add or Delete Update Type:

30	1 char	ETX	End of message; do not fill in following fields

For Modify Update Type:

List only the modified values in the following fields. Put a single blank in the fields not modified.

30	10 chars		System ID where data stored
31	1 char	LF	Field delimiter

32	3 chars		DBMS name
33	1 char	LF	Field delimiter
34	1 char		DBMS type
35	1 char	LF	Field delimiter
36	15 chars		Database name*
37	1 char	LF	Field delimiter
38	15 chars		Local relation name*
39	1 char	LF	Field delimiter
40	15 chars		Local attribute name*
41	1 char	LF	Field delimiter
42	1 digit		Index Code
43	1 char	LF	Field delimiter
44	2 digits		Replication code
45	1 char	LF	Field delimiter
46	1 char	ETX	End of message

* If the length of a name is less than its maximum size, a LF is placed immediately after the names without any padded blanks before the LF.

Local Query Request Message and Remote Query Request Message

Field No.	Field Definition	Value	Description
1	1 char	STX	Start of message
2	3 chars		Message type
		LQR	Local Query Request Message
		RQR	Remote Query Request Message
3	1 char	LF	Field delimiter
4	10 chars		System ID at destination computer
5	1 char	LF	Field delimiter
6	10 chars		System ID at source computer
7	1 char	LF	Field delimiter
8	4 chars		Unique process ID
9	1 char	LF	Field delimiter
10	10 chars		Time stamp (HH:MM:SS.T)
11	1 char	LF	Field delimiter
12	10 chars		Password
13	1 char	LF	Field delimiter
14	Varies		Query
15	1 char	ETX	End of message

Local Query Results and Remote Query Results

Field No.	Field Definition	Value	Description
1	1 char	STX	Start of message
2	3 chars		Message type
		LQM	Local Query Message
		RQM	Remote Query Message
3	1 char	LF	Field delimiter
4	10 chars		System ID at destination computer
5	1 char	LF	Field delimiter
6	10 chars		System ID at source computer
7	1 char	LF	Field delimiter
8	4 chars		Unique process ID
9	1 char	LF	Field delimiter
10	10 chars		Time stamp (HH:MM:SS.T)
11	1 char	LF	Field delimiter
12	Varies		Query Results
13	1 char	ETX	End of message

APPENDIX F

PUBLICATION ARTICLE

Report on

DESIGN AND IMPLEMENTATION OF A

CENTRALIZED DATA DIRECTORY SYSTEM FOR A

DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Introduction

Many organizations store the data used in their various computer programs in a database. This allows them to centralize the information so that it is easier to retrieve and change the data. A centralized database management system (DBMS) consists of software residing on one computer which structures the data and manipulates it so that many application programs can access it. On the other hand, a distributed database management system (DDBMS) manipulates separate databases stored on host computers which are linked by a network. Distribution is transparent to the user so he can access any data in the system without having to know where it is stored. A directory system, rather than the user, keeps track of the data locations.

Imker designed a DDBMS using the three types of directories (Imker, 1982:63-79). A centralized directory, called a centralized network data directory (CNDD), is stored only on one system. It contains a conceptual view of the data entities in all the DBMSs. An extended directory, called an extended centralized network data directory (ECNDD) is a small local version of the CNDD. Whenever a site requests the

location of data from the CNDD, the local site copies the information into its ECNDD so it does not have to ask the CNDD for the location again. The third type of directory is the local network data directory (LNDD). This is a directory of the data in the site's local DBMS.

Problem

This research, done at the Air Force Institute of Technology (AFIT), further refined the DDBMS design of Capt John G. Boeckman (Boeckman, 1984). The objectives of this research were to:

a) Design, implement, and initialize the centralized data directory (CNDD)

b) Implement the software to request CNDD data

c) Implement the processing to retrieve data locations stored in the CNDD

This effort followed the generally accepted life cycle method, namely: a) requirements analysis, b) detailed design, c) implementation, and d) integration testing.

Analysis of Requirements

The central site has the following functions to control the directory system (Boeckman, 1984:20-21):

1. Initialize the DDBMS
2. Service the Centralized Network Data Directory (CNDD) site requests
3. Send updates to Extended Centralized Network

Data Directories (ECNDD) which contain copies of data changed in the CNDD.

Initialize the DDBMS. Initialization of the DDBMS occurs when the system starts up. Different procedures occur depending on whether the site is the central site or not. If it is the central site, the software initializes the CNDD, queries the other sites, evaluates their responses, and sends a startup message to all the sites participating in the DDBMS. If the site is not the central site, software initializes the site's database and responds to the central site's query.

Update the ECNDDs. The central site is also involved with all directory updates. If data changes at a site and affects its local directory (LNDD), the central site must update the CNDD. The central site also must determine what sites had requested the locations of the data that changed. Then the central site sends changes to these sites so they can change their ECNDD.

Service Requests at Central Site. Just as with the other sites, the central site must process several types of requests. They may be either CNDD data location requests, CNDD updates, or pending update requests.

To service CNDD data location requests, the central site searches the CNDD and returns all the locations of the data requested.

To service CNDD updates due to LNDD updates, the CNDD site receives the CNDD updates from another site and matches the received data against the data in the CNDD. Next it updates the CNDD and sends an update acknowledgement message to the sending site. Then it sends updates to the ECNDDs which also have the data. Finally, the central site receives an ECNDD update acknowledgement message from the other sites which received ECNDD updates.

The last CNDD request type is servicing pending update requests. For this request, the central site adds information to the pending update file of an inactive site. This file stores all changes users make to data stored in sites that are temporarily disconnected from the DDBMS. Also, the central site sends the results of the update back to the site which originated the pending update request.

General Content of Data Directories

Jones (Jones, 1984:149-153) presented what information a data dictionary should contain when using a global relational data model in a heterogenous DDBMS. It included information about the databases in the system, what relations were stored in each database, the attributes of each relation and other information needed to map--or translate--from the global language to a local database definition language.

Based on Jones' research, the following information was included in the CNDD and ECNDD:

a. Site identification of source (identifies the

network address of the site)

b. Host computer (e.g. UNIX VAX)

c. DB name (e.g. AFIT, Demo, etc.)

d. Global relation name ("Global" name is a common name for possibly several local relations with different names stored in separate databases. A global relation identification was not needed because the global relation name must uniquely identify the relation.)

e. Relation replication code (specifies whether data is duplicated in several databases and how the data is partitioned)

f. Global attribute identification

g. Global attribute name

h. Local relation identification ("Local" relation is a relation stored at a local database. If the local DBMS was a network or hierarchical type DBMS, the entity was translated to a relational type before storing it in the directory. In a concurrent research effort, Mahoney (Mahoney, 1985) stored the mapping information needed for this translation elsewhere.)

i. Local relation name

j. Local attribute identification

k. Local attribute name

In addition to Jones' requirements, the fcllowing items were necessary to implement the directory system:

a. Access code (prevents CNDD from releasing data

that is being updated)

b. DBMS name (e.g. DBTG, INGRES, dBASE II, Total)

c. DBMS type (e.g. hierarchical, relational or network)

d. Local relation index code (specifies whether the relation is indexed on a particular attribute).

As for the LNDDs, they contain the information above except their own site identification and site name. They also contain other information needed to map data definitions from one type of DBMS to another. The LNDD should store the mapping information because the processing does not need the information until just before sending a query to the host database. Therefore, when a site receives a query to send to its host DBMS, the processing uses the information to convert from the global relational data descriptions to those used in the host database.

Detailed Design of Servicing CNDD Site Requests

The structure chart in Figure F-1 shows three different kinds of requests the CNDD site processes: data location requests, CNDD updates and pending update requests.

The following section explains in detail how the CNDD site services data location requests. The next section explains the conceptual procedures for updating the CNDD. This paper does not explain the detailed design of servicing pending update requests. However, Boeckman completed a general design in his study (Boeckman, 1985:34).

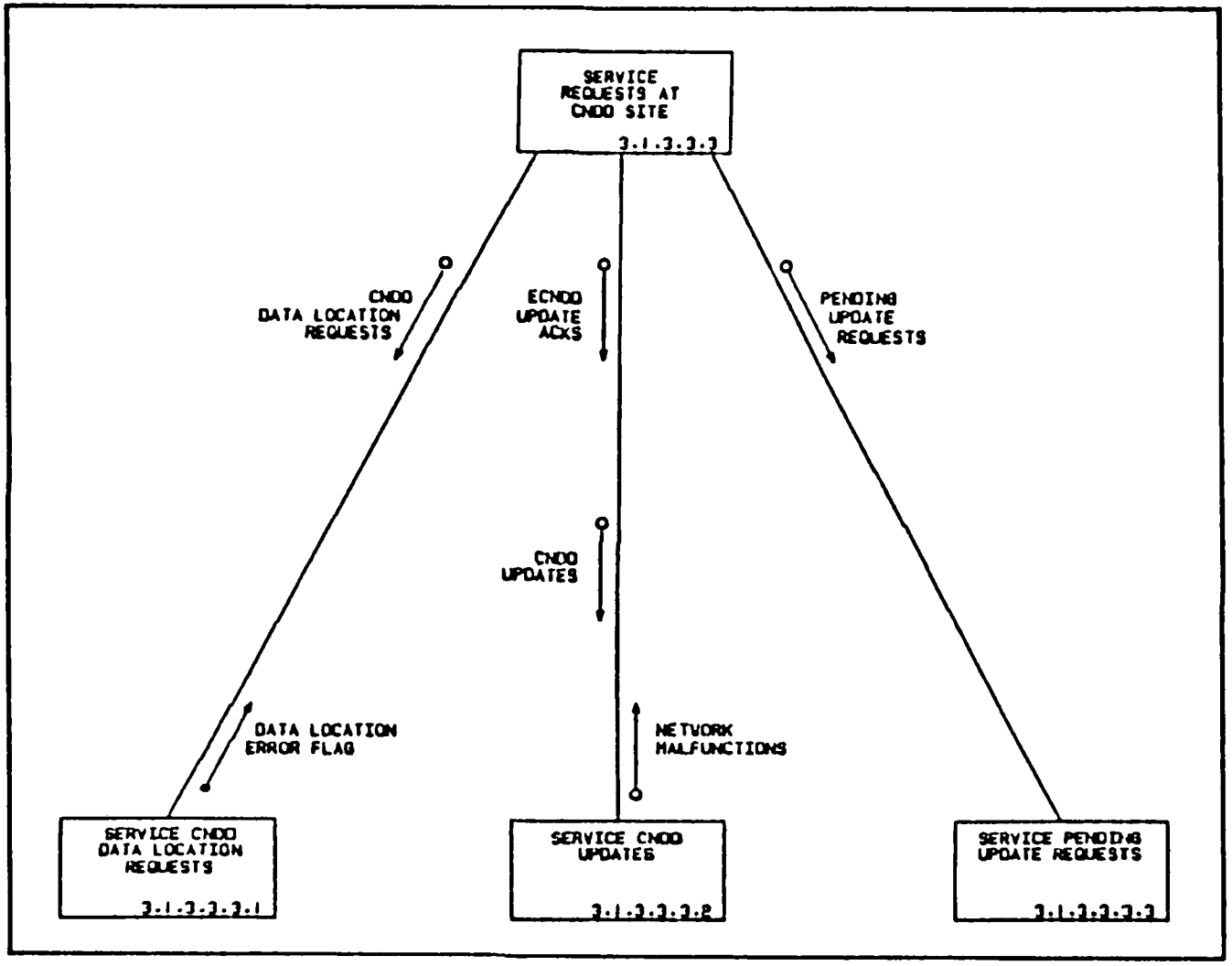


Figure F-1. Service Requests at CNDD Site

Data Location Requests. For data location requests, the central site first verifies whether the CNDD Data Location Request message contains the correct password in order to access the CNDD. The software then extracts information from the request message in order to build a standard header for the results message, which will contain all of the data location information retrieved from the CNDD.

Since the user's query is written in a relational data manipulation language, the query includes names of relations and attributes. From the user's viewpoint these relation and attribute names are global names. In other words, they are names used at the highest conceptual level with which the user is familiar. In contrast, the local relation and attribute names are those names used in a specific host database. The local names may be different from the global names or the same as the global names.

Figure F-2 shows four high-level steps of servicing a CNDD data location request. First a module gets the request type and a global relation name from the request message. This step was added to Boeckman's design because of the decision to combine several request types into one message format. Next, the CNDD processing extracts the data locations of one relation at a time. Then it reformats the information returned from the CNDD into the CNDD Data Location Results message. These three steps continue until the CNDD has found the locations of all the relations and attri-

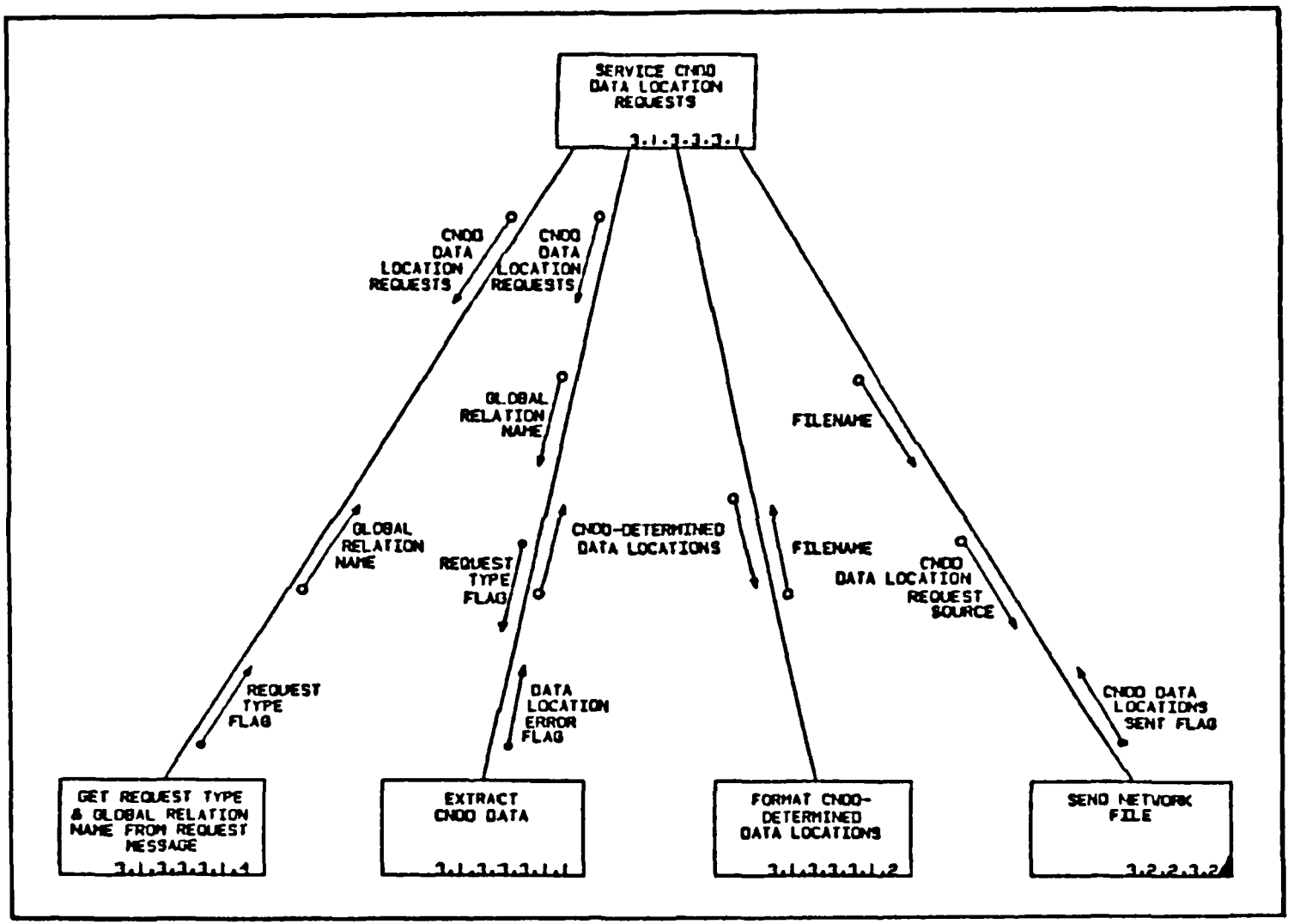


Figure F-2. Service CNDD Data Location Requests

butes in the request message. Finally, the CNDD site sends the results message to the requesting site.

CNDD Update Requests. Another function of the CNDD is to service CNDD update requests. The following is a conceptual idea of how to process the update. Part of the process must be manual because the central database administrator (DBA) responsible for controlling the update may have to make some decisions before the update can proceed. For example, if a new relation is added at a site, someone has to decide to which global relation(s) the local relation belongs. He also has to match the local attributes within the new local relation with the global attributes within the global relation. To explain this process, Figure F-3 shows the upper-level modules required to service this request.

First, when the CNDD receives an update message from a site, it locks the access to the global relation's data. This prevents the CNDD from sending to a requesting site any data location information on the global relation that is not current.

Second, the CNDD site services the updates to the CNDD sent from sites that intend to update their LNDDs. The CNDD site software displays a message on the central site's terminal, explaining the changes to be made and writes the same information to a file. This allows the central DBA to review the information off-line. After making the necessary decisions, like global relation-local relation mappings, the

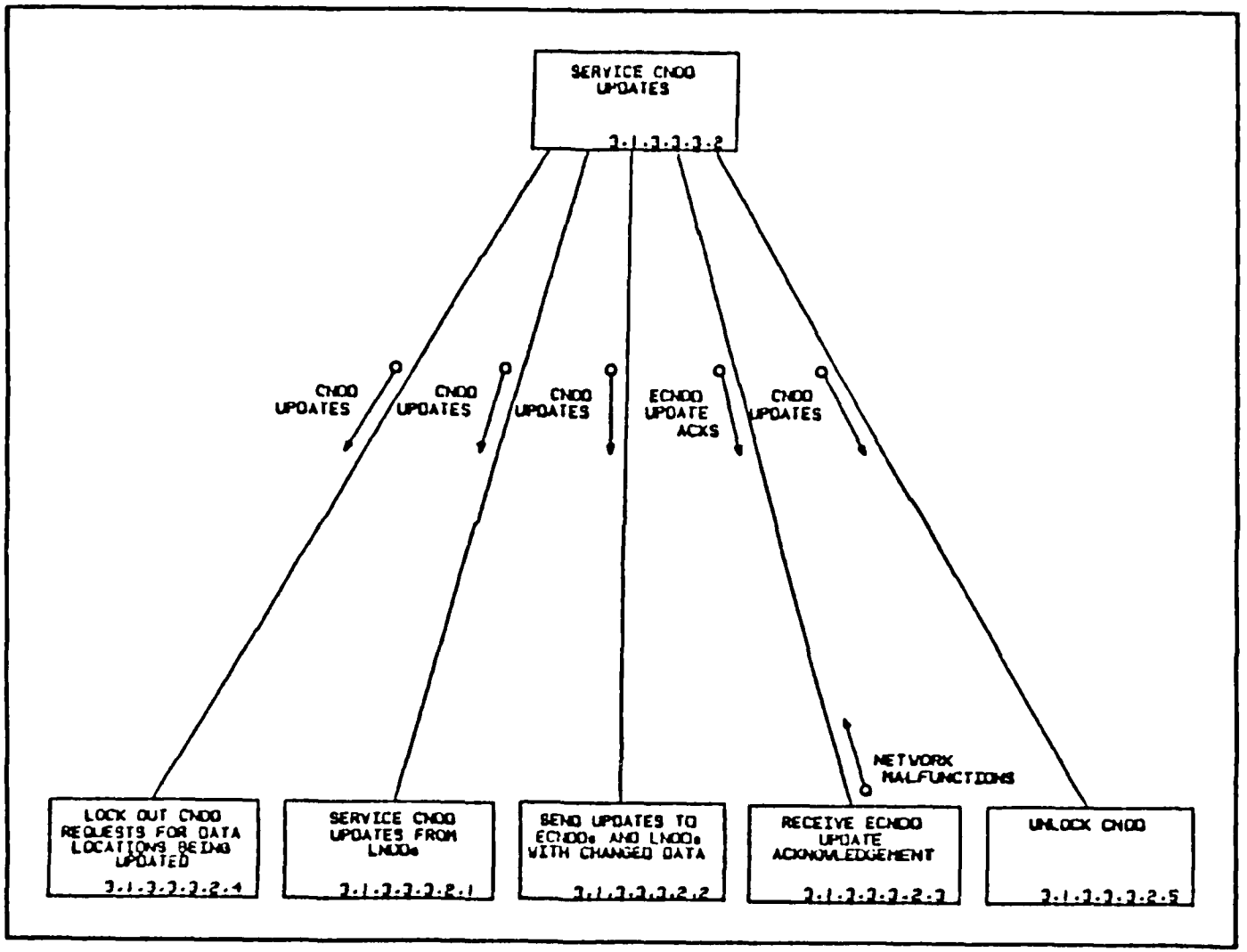


Figure F-3. Service CNDD Updates

central DBA manually changes the CNDD when the system is off-line. He also marks that the update is completed in the file that contained the information on the update.

When the DDBMS comes back on-line, part of the CNDD initialization processing checks this file. If there are CNDD updates marked as completed in the file, the CNDD software checks which ECNDDs and LNDDs must be changed because of the data just changed in the CNDD.

In the third major step, the CNDD sends updates to ECNDDs and LNDDs which must be changed. When the site which originally sent the update to the CNDD receives the LNDD update message from the CNDD site, it can finally update its LNDD.

The CNDD site waits for an acknowledgment message from the ECNDDs in the fourth step. When the CNDD site receives all the ECNDD acknowledgement messages it expects, it unlocks the CNDD in the fifth and final step.

Partial Implementation

This project implemented the same DDBMS detailed design described in Boeckman's thesis. The implementation followed a top-down programming approach. Because of the time constraint and scope of this research, not all the DDBMS was implemented. Since the centralized network data directory system (CNDD) was the main thrust of this effort, this phase completed all of the processing to make a request for data from the CNDD and to get the data locations from the CNDD.

Implemented Architecture. Figure F-4 shows the architectural topology of the hardware used in this implementation. The DDBMS system consisted of two LSI-11 microcomputers and one Z-80-based S-100 bus microcomputer. The LSI-11 computers were identified as System L and System S in the AFIT Digital Engineering Laboratory.

System L acted as the CNDD site in the DDBMS. Because of memory limitations, System L only contained the DDBMS software necessary to process CNDD site requests. It did not process queries to the distributed databases. It connected to a host S-100 microcomputer, which executed the dBASE II DBMS to load, update and access data in the CNDD.

The other LSI-11 computer, System S, was a remote DDBMS site which executed the software to handle the DDBMS queries and create data location requests for the CNDD site. Although the computers were nodes on the LSINET, because of memory sizing problems, these LSI-11 computers were unable to contain the network operating system (NETOS) used for the LSI-11 computers to communicate between. NETOS required 34K, the DDBMS software needed 40K and the CNDD processing used 36K. In order to link the DDBMS with a network, therefore, the three programs must be hosted on different computers.

Implementation of CNDD. The CNDD was implemented using a host DBMS. Due to the memory restrictions and the scope of the thesis, only the data location requests were processed at the CNDD site.

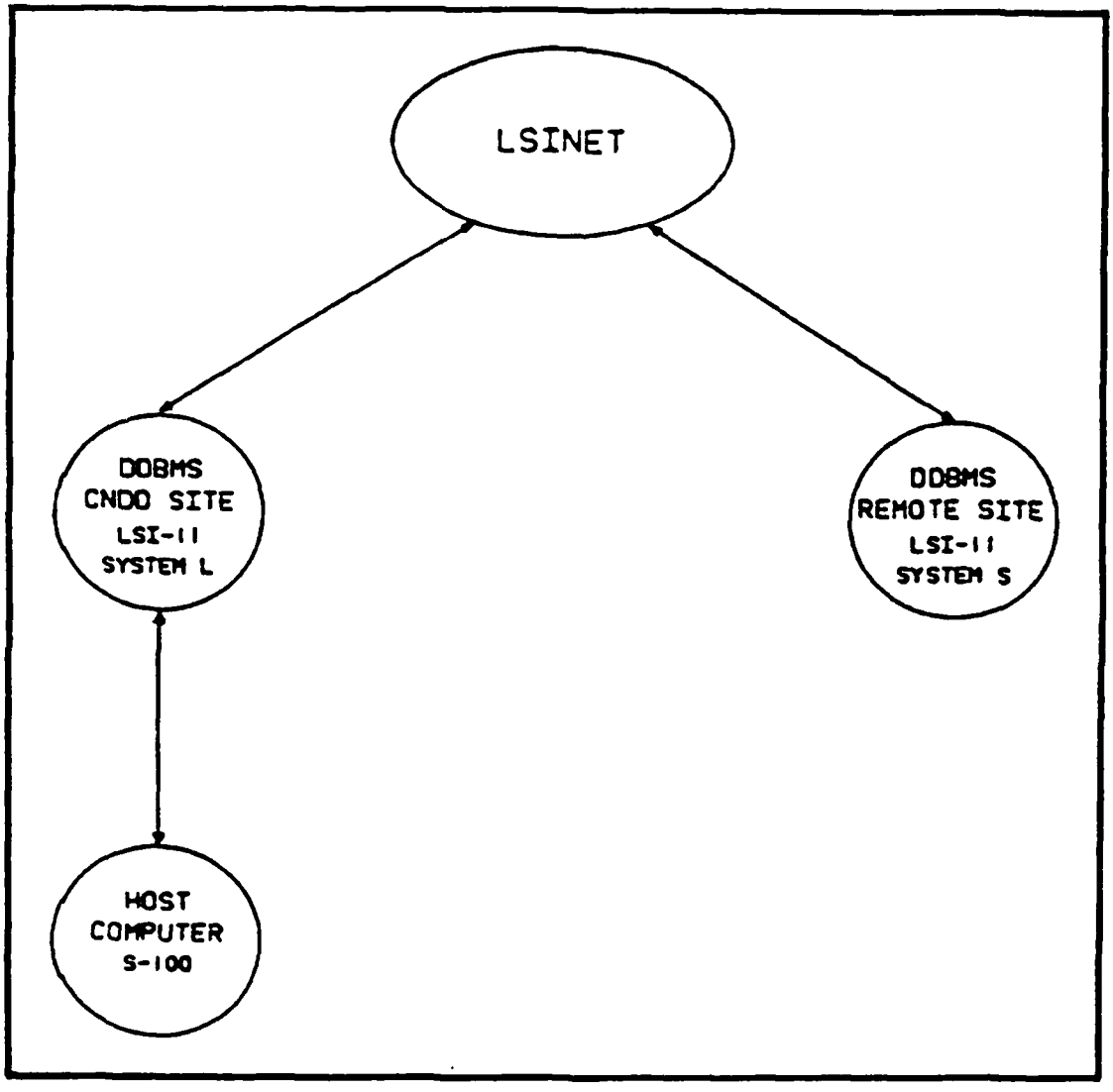


Figure F-4. Implemented Architecture

The CNDD data was originally organized into the relations shown in Figure F-5a. These original relations were all normalized to the third normal form. However, many of these relations were combined to make the CNDD processing more efficient. Figure F-5b shows the final six CNDD relations formed from those in Figure F-5a and loaded into a database with the dBASE II relational DBMS. DBASE II command files contained relational algebra operations to retrieve the data locations of the global attributes within a global relation. The CNDD processing then started the execution of dBASE II on the host computer, which in turn executed command files to get the information from the CNDD database.

System Integration Testing

In this phase of the project all the software modules implemented were integrated and tested to determine whether they performed together correctly. As the main objective, the testing evaluated the process of requesting and extracting data locations from the CNDD. This involved breaking the testing into two steps: 1) constructing a CNDD Data Location Request message, and 2) extracting the information requested from the CNDD and constructing a CNDD Data Location Results message.

CNDD Test Data. Two test databases were constructed on different host computers, each of which executed a relational DBMS. A dBASE II DBMS ran on an S-100 microcomputer, and an INGRES DBMS ran on a VAX-11/780 minicomputer. Although the

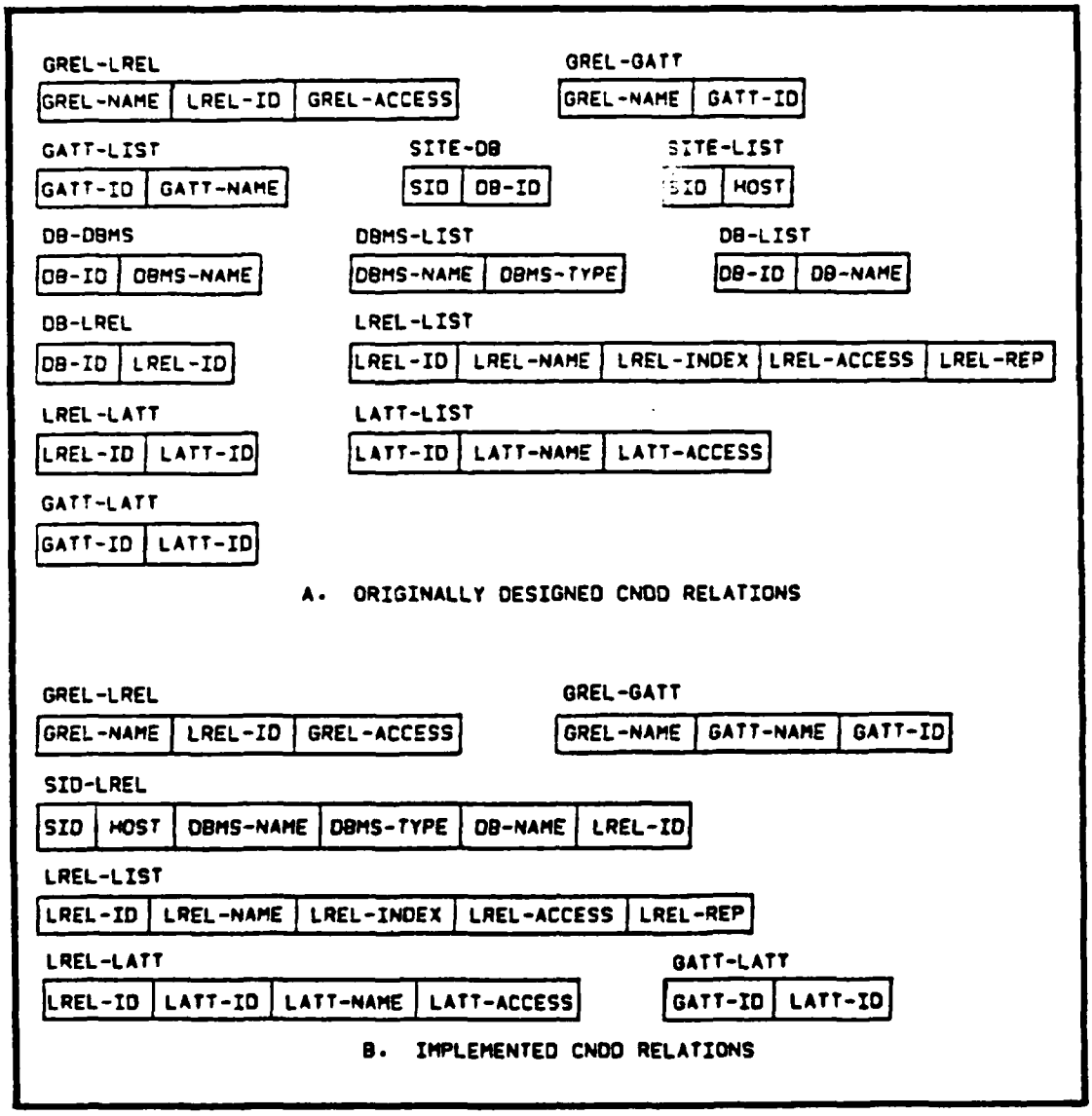


Figure F-5. CNDD Relations

```
Query #1:  SELECT ALL FROM parts
           WHERE (city = 'Chicago') GIVING newrel

Query #2:  JOIN parts, receipt
           WHERE pnum = pnum GIVING newrel

Query #3:  PROJECT suppliers OVER snum, sname, bad
           GIVING newrel

Query #4:  SELECT ALL FROM inventory
           WHERE (pnum = 'PI') GIVING newrel
```

Figure F-6. Test Queries

tests did not access these databases through queries, the locations of all the data were stored in the CNDD.

Remote Site Processing. During the tests the remote site program called "DDBMS" only handled a query up to the point of creating a message that requests data locations to send to the CNDD site. The site did not send the message to the CNDD nor did it send the query to the host databases. The test procedures were as follows.

First, test queries were created with a text editor and stored in ASCII files. The tests used the queries shown in Figure F-6. These queries requested data stored only in one database or in both databases. Also, the third query included an attribute "bad" which was not part of the global relation. The last query required data from the CNDD that was locked, to simulate the data in the CNDD associated with a relation being updated.

Finally, the remote site software created a file which contained a CNDD Data Location Request message. For example, the messages for queries #1, 2 and 4 requested the locations of all the global attributes within the global relations "parts", "parts", and "inventory", respectively. The message for query #2 did not include the relation "receipt" because its location was simulated to be in the LNDD or ECNDD. In contrast, the request message for query #3 asked for the locations of only the global attributes "snum", "sname" and "bad" within the relation "suppliers". Test results verified that the request message for each query was built correctly.

CNDD Site Processing. Once the remote site built the CNDD Data Location Request messages, tests checked whether the CNDD site processing extracted the data correctly. The tests simulated receiving a message from the network by reading a file. Four files, built with a text editor, contained the same request messages that the remote site constructed during its four test runs.

The program "CNDD" on System L executed four different runs to process each Data Location Request message stored in a file. The software accessed the CNDD and retrieved all the information requested. At the end of the processing, the CNDD site program created a file with the formatted CNDD Data Results message. After each run, a text editor was used to check that each results message contained the correct format and the required data locations.

Results and Conclusions

This project accomplished the main goal of designing the CNDD, implementing it on one of the DDBMS sites, and implementing the software which creates and processes requests for data locations stored in the CNDD. The integration testing period proved the implemented code worked according to the system requirements and design.

Unfortunately, the DDBMS sites were not connected to a network so that messages could be passed from one site to another. Both the DDBMS software implemented so far and the operating system (NETOS) for the LSINET local area network would not operate on a single LSI-11 microcomputer together. The computer's operating system could not execute all the software in the memory allotted for the program. Consequently, resolving this problem should be the first priority in any future development of the AFIT DDBMS project.

In addition, the thesis described the design of the network messages and the process to update the CNDD, but it did not implement the process. The detailed design also specified the data contents of the LNDD and the ECNDD. However, the project did not implement them nor develop the software which checks for data locations in these local directories.

Follow-on Research

Future work on the AFIT DDBMS should concentrate on connecting the DDBMS in a network, first and foremost. Other

projects include implementing the LNDD and ECNDD, updating the ECNDD from CNDD results, initializing the DDBMS, updating the CNDD, processing pending updates, optimizing the query partitioning and implementing message queues. In order to implement all these capabilities, though, the AFIT Digital Engineering Laboratory will need a multi-processing operating system with virtual memory addressing for its network computers. The DDBMS design is just too big to implement on the current microprocessor equipment. For example, the DDBMS software could be rehosted on the Intel Hypercube, a multiprocessor computer. Continuing research in these areas will some day make distributed database management systems a practical reality.

Bibliography

Allen, Frank W. and others. "The Integrated Dictionary/ Directory System." ACM Computing Surveys, 14: 245-275 (June 1982).

Boeckman, Capt John G. Design and Implementation of the Digital Engineering Laboratory Distributed Database Management System. MS thesis, GCS/ENG/84D-5. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.

Ceri, Stefano and Guiseppe Pelagatti. Distributed Databases, Principle and Systems. New York: Mcgraw-Hill Book Co, 1984.

Chu, Wesley W. "Performance of File Directory Systems for Data Bases in Star and Distributed Networks," American Federation of Information Processing Societies Conference Proceedings, 45: 577-587 (June 1976).

Date, C. J. An Introduction to Database Systems. Reading MA: Addison-Wesley Publishing Company, 1982.

Durell, W. "Disorder to Discipline Via the Data Dictionary," J. Syst. Manage., 34; no. 5: 12-19 (May 1983).

Garcia-Luna-Aceves, J. J. and F. F. Kuo. "A Hierarchical Architecture for Computer-based Message Systems," IEEE Transactions on Communications, 30 (1): 37-45 (Jan 1982).

Hartrum, Thomas C. Lecture materials on the AFIT Digital Engineering Laboratory LSINET distributed in EE 6.90, Software Systems Laboratory. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, July 1985.

Imker, Capt Eric F. Design of a Distributed Database Management System For Use in the AFIT Digital Engineering Laboratory. MS thesis, GCS/EE/82D-21. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.

Jones, 2Lt Anthony J. Analysis and Specification of a Universal Data Model for Distributed Database Systems. MS thesis, GCS/ENG/84D-11. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1984.

Lefkovits, Henry C. Data Dictionary Systems. Wellesley MA: Q. E. D. Information Sciences, Inc., 1977.

Leong-Hong, Belkis W. and Bernard K. Plagman. Data Dictionary/Directory Systems. New York: John Wiley & Sons, Inc., 1982.

Mahoney, Capt. Kevin H. The Design and Implementation of a Relational to Network Query Translator for a Distributed Database Management System. MS thesis, GCS/ENG/85-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB Oh, December 1985.

Peebles, Richard and Eric Manning. "System Architecture for Distributed Data Management," Tutorial: Centralized and Distributed Data Base Systems, 352. New York: IEEE Computer Society, 1979.

Peters, Lawrence J. Software Design: Methods and Techniques. New York: Yourdin Press, 1981.

Roth, 2Lt. Mark A. The Design and Implementation of a Pedagogical Relational Database System. MS thesis, GCS/EE/79-14. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1979.

Rowe, Capt. Janice F. A Network Monitoring Facility for a Distributed Database Management System. MS thesis, GCS/ENG/85-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.

Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs NJ: Prentice-Hall, Inc., 1981.

Uhrowczik, P. P. "Data Dictionary/Directories," IBM System Journals: 12, November 4 (1973).

Ullman, Jeffrey D. Principles of Database Systems, second edition. Rockville MA: Computer Science Press, 1982.

VITA

Captain James A. Wedertz was born on 12 April 1951 in San Francisco, California. He graduated from high school in San Mateo, California, in 1965 and attended Brigham Young University in Utah from which he received the degree of Bachelor of Science in Computer Science in December 1975. As a distinguished graduate, he received a commission in the USAF through the ROTC program. He was employed as a systems programmer at the Sperry Univac Company, Salt Lake City, Utah, until called to active duty in June 1976. He served as a systems analyst at the SAGE Programming Agency, Luke AFB, Arizona, and as a software configuration manager at HQ NORAD, Colorado Springs, Colorado. He then served as a computer systems staff officer in the Personnel Exchange Program at the Venezuelan AF headquarters, Caracas, Venezuela, until entering the School of Engineering, Air Force Institute of Technology, in May 1984.

Permanent address: 2311 South Norfolk Street
San Mateo, California 94403

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A163 843

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/ENG/85D-24	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.; PROJECT NO.; TASK NO.; WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): James A. Wedertz, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1985 December; 15. PAGE COUNT: 149	
16. SUPPLEMENTARY NOTATION	

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR	Data Bases, Data Base Management Systems, Distributed Data Base Management Systems, Networks, Directories
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: DESIGN AND IMPLEMENTATION OF A CENTRALIZED DATA DIRECTORY FOR A DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER 16 JAN 86
Dean for Research and Professional Development
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

Thesis Chairman: Dr. Thomas C. Hartrum
Assistant Professor of Electrical Engineering

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Dr. Thomas C. Hartrum	513-255-2024	AFIT/ENG

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This study refined and implemented a design of a centralized data directory for a distributed database management system (DDBMS) begun in a previous study for use in the AFIT Digital Engineering Laboratory. This directory contains information about all the data stored in the distributed databases. By following the life cycle programming method to develop the system, this project completed a requirements analysis, detailed design and implementation of the directory as well as a partial implementation of the DDBMS to test the operation of the centralized data directory.

The requirements analysis outlined the functions of the central site, which contained the centralized directory. This project used Structured Analysis Design Technique (SADT) diagrams to document the central site's functions. These included initializing the DDBMS, updating the centralized directory, sending changes to other local directories at the remote sites, reconfiguring the DDBMS and servicing requests for information in the directory.

Next, the project refined the detailed design of the directory processing and depicted the functional decomposition in structure charts. The following step implemented on two microcomputers only those modules necessary to show the centralized directory worked. Tests verified that one DDBMS node which received a query could request and receive location information from the other node.

END

FILMED

3-86

DTIC